**Technische Fakultät**
Albert-Ludwigs-Universität, Freiburg
Lehrstuhl für Kommunikationssysteme
Prof. Dr. Gerhard Schneider

Masterarbeit

# Emulatoren-Testing für die digitale Langzeitarchivierung

3. März 2011

Nana Tchayep Achille
Matr.-Nr.: 1810628

betreut durch
Dr. Dirk von Suchodoletz
Klaus Rechert

## Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, 3. März 2011 Unterschrift

**Danksagung**

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Erstellung dieser Masterarbeit unterstützt haben. Insbesondere gilt dieser Dank Dr. Dirk von Suchodoletz und Klaus Rechert für die konstruktiven Anregungen und die angenehme Arbeitsatmosphäre sowie Prof. Dr. G. Schneider für die Möglichkeit am Lehrstuhl für Kommunikationssysteme meine Masterarbeit anfertigen zu dürfen.

Bei Konrad und Volker möchte ich mich für die Unterstützung und die konstruktive Kritik zur Ausarbeitung bedanken. Zudem gilt mein Dank allen weiteren Personen in meinem Umfeld für ihre Ermutigungen und moralische Unterstützung.

Zu guter Letzt möchte ich mich sehr herzlich bei meinen Eltern in Kamerun bedanken, die stets hinter mir gestanden sind. Dieser Dank gilt auch meinen Geschwistern Aurelien, Serge Carine, Guy und Nelly.

# Inhaltsverzeichnis

# 1 Einleitung

Die technologischen und wissenschaftlichen Fortschritte haben die Gesellschaftsgewohnheiten verändert. Diese Veränderungen werden zum größten Teil durch die rasche Verbreitung von Informationen, die vom rasanten Anstieg von Informationsquellen (Tablet-PC, Fernseher, Internet, Smartphones) verursacht wird, hervorgerufen. In diesem Informationsfluss stehen digitale Objekte im Mittelpunkt. Die Erstellung solcher Objekte kann relativ leicht erfolgen, aber deren Langzeitarchivierung mit dem Ziel die enthaltenen Informationen für die Nachwelt aufzubewahren, stellt die wissenschaftliche Welt vor neuartige Herausforderungen.

Die Langzeitarchivierung digitaler Objekte benötigt andere Konzepte als die Langzeitarchivierung klassischer Objekte wie Papier, Tontafeln, Papyrus, Pergament, Zeitschriften und natürlich auch Bücher, denn dabei geht es nicht nur um eine bloße Sicherung der Objekte selbst, sondern auch um ihre dauerhafte Verfügbarkeitsmachung [8]. Die eigentlichen enthaltenen Informationen lassen sich generell in einer klar definierten Kontext-Umgebung abrufen. Zum Beispiel ist die Verarbeitung eines *.docx*-Dokuments erst ab der Version von *Microsoft Office 2007*[1] ohne Zusatzprogramme möglich. Die Verarbeitung eines *.sam*-Dokuments erfolgt dagegen unter *Lotus Amipro*, einer veralteten Windows-Anwendung.

Die Kontext-Umgebung an sich, setzt sich aus Software und Hardware zusammen. Mit der technologischen Entwicklung werden sie obsolet und diese Tatsache lässt die Frage über die zukünftige Betrachtung von digitalen Objekten offen [7]. Deshalb liegt die eigentliche Herausforderung bei der Langzeitarchivierung digitaler Objekte darin, eine passende Kontext-Umgebung für das jeweilige digitale Objekt dauerhaft bereitzustellen. Zur Erfüllung dieser Mammutaufgabe werden die Technik der Emulation [8; 31; 33; 14] und der Migration eingesetzt.

Da die Kontext-Umgebung veränderbar ist, wird dadurch die Betrachtung eines digitalen Objekts in seiner ursprünglichen Umgebung erschwert. Der Einsatz von Migrationsstrategien bietet die Möglichkeit, das Objekt so umzuwandeln, dass es in einer

[1] Eine Bezeichnung für eine Office-Produktlinie von Microsoft. Es wurde am 30. Januar 2007 als Ersatz für *Microsoft Office 2003* erstellt. http://office.microsoft.com

aktuellen digitalen Umgebung wieder zugreifbar wird. Durch diese Umwandlung ist mit Informations-, Integritäts- und Authentifizierungsverlust zu rechnen, besonders, wenn es sich um große Datenmenge handelt [31, S. 15]. Außerdem lässt sich die Migrationsstrategie nur auf statische Objekte (z.B.: Text, Bilder) anwenden, denn die dynamischen Objekte (Anwendungsprogramme, Betriebssysteme) stellen keine erforschten und erprobten Herausforderungen dar [31].

Aus diesen Gründen soll die Strategie der Emulation angewendet werden. Im Gegensatz zur Migration, agiert die Emulation nicht auf das Objekt selbst, sondern auf die Kontext-Umgebung, in der das Objekt ursprünglich erstellt wurde. Die Methode der Emulation sichert ein funktionelles Abbild eines Systems. Der Emulator versucht das Verhalten bzw. die Funktionsweise eines Systems so zu reproduzieren, als ob dieses real wäre. Die erste Umsetzung einer Emulation wurde im Jahre 1962 in Frankreich von IBM durchgeführt. Es ging um die Kompatibilitätsprüfung von neuen Produkten zu ihren Vorgängern. Im Jahr 1978 wurde der erste Spielkonsoleemulator[2] von der Firma Coleco[3] entwickelt.

## 1.1 Motivation

Im Kontext der Langzeitarchivierung wurde die Methode der Emulationsstrategie erst im Jahr 1999 von Rothenberg Jeff[4] vorgeschlagen. Mit der Verwendung dieser Strategien soll die Langzeitarchivierung die Erhaltung der ursprünglischen technischen Umgebung (Anwendung, Betriebssystem und Hardware) der digitalen Objekten sichern [31; 33], damit das betroffene Objekt weiterhin betrachtet oder bearbeitet wird.

Der Erfolg der Emulationsstrategien hängt von eingesetzten Emulatoren ab. Die Emulatoren sind Software (QEMU[5], Bochs[6], MESS[7], Dioscuri[8]) und aus der Sicht der Archiv-Verwaltung sind sie nicht von digitalen Objekten zu unterscheiden; Das heißt,

[2] Es handelte sich um einen Adapter für den Atari 2600. Atari war der größte Lieferant der klassischen Videospiele und Zubehör. https://www.atari2600.com

[3] Ein Hersteller der Unterhaltungselektronik, der Spielkonsole wie *ColecoVision* hergestellt hat. http://www.coleco.com

[4] Siehe hierzu [14]. Homepage des Autors http://www.panix.com/~jeffr/Prof/prof.html

[5] Eine freie virtuelle Maschine, die es erlaubt, auf ein schon installiertes Betriebssystem eine oder mehrere Betriebssysteme zu emulieren, http://wiki.qemu.org

[6] Das ist ein freier x86- und AMD64-Emulator und Debugger, http://bochs.sourceforge.net

[7] Der Multi Emulator Super System ist ein Emulator für zahlreiche Spielkonsole und andere Informatiksystem, http://www.mess.org

[8] Ein java x86-Hardware-Emulator, http://dioscuri.sourceforge.net

sie sind ebenfalls mit einer Kontext-Umgebung verbunden. Sollte diese Kontext-Umgebung obsolet werden, wäre der Emulator nicht mehr ausführbar und somit wäre der Zugriff auf die bisherigen emulierten Umgebungen bzw. die entsprechenden digitalen Objekte nicht mehr gewährleistet. Daher sollen die Emulatoren aktuali-

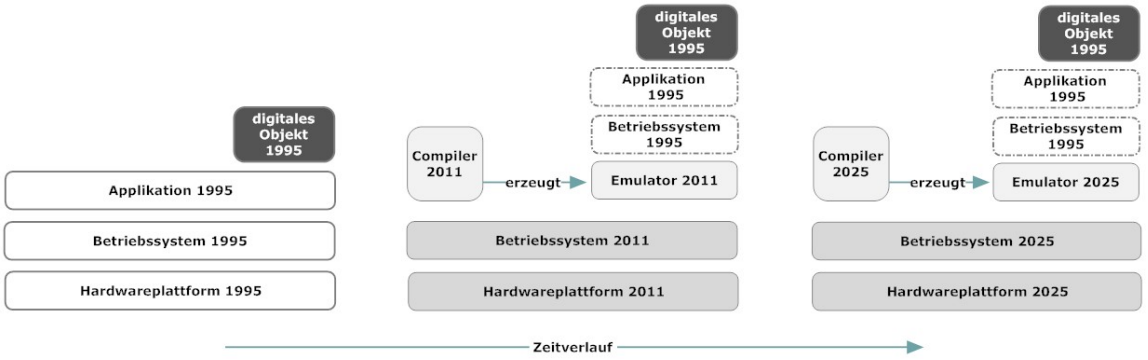


Abbildung 1.1: Emulatoren im Wandel der Zeit

siert werden (s. Abbildung 1.1), um auf die heutige bzw. jeweils aktuelle Software-, Hardware-Landschaft angepasst zu werden oder um mit neuen Features bestückt zu werden und außerdem noch um schon eventuelle existierende Fehler zu beheben. Im Kontext der Langzeitarchivierung lässt die Aktualisierung von Emulatoren einige Punkte offen. Die aktualisierten Emulatoren können nicht unmittelbar ins Archiv-System übernommen werden. Bevor die neuen Emulatoren die alten ablösen, muss sichergestellt werden, dass die bisherigen Emulatoren-Funktionalitäten dadurch nicht verloren gegangen sind.

## 1.2 Problemstellung

Die Aktualisierung einer Software bedarf die Durchführung bestimmter Tests, um sicherzustellen, dass die Software ihre Anforderungen erfüllt. In dieser Hinsicht behauptete den Informatikpionier Edsger W. Dijkstra[9]:

> *I shall not deny that the construction of these testing programs has been a major intellectual effort: to convince oneself that one has not overlooked "a relevant state" and to convince oneself that the testing programs generate*

[9] Der Mathematiker und Informatiker war der Wegbereiter strukturierter Programmierung. Siehe hierzu: http://www.cs.utexas.edu/users/EWD

*them all is no simple matter. The encouraging thing is that (as far as we know!) it could be done.*

Jeder Fehler, der vor dem Einsatz eines neuen Produkts nicht entdeckt wird, kann unwiderrufliche Folgen haben. Folgende Vorfälle machen dies noch deutlicher:

- In der Weihnachtszeit im Jahre 1994 veröffentlichte *The Walt Disney Company*[10] seinen ersten Multimedia CD-ROM-Zeichentrickfilm für Kinder: "*Der König der Löwen*". Mit dem Verkauf des CDs erzielten sie große Gewinne. Keiner konnte jedoch ahnen, was danach folgen würde. Am 26. Dezember desselben Jahres beschwerten sich tausende von Eltern mit weinenden Kindern beim Disney's Support-Team , weil sie die Software auf ihren Maschinen nicht ausführen konnten[11].

- Am 3. Dezember 1999 verschwand die Raumsonde *Mars Polar Lander*[12] während eines Landeablaufes auf dem Mars. Die Ursache dieses Verlustes war eine verfrühte Abschaltung der Bremstriebswerke. Beim Ausklappen und Arretieren der Landebeine wurde ein Ruck auf den Landesensor übertragen, der sich am Ende eines Beines befand und eigentlich Bodenkontakt melden sollte[13]. Dabei wurde unerwartet ein einziges Bit nicht gesetzt. Dieser Fehler ist nicht nur auf finanzielle Probleme oder auf die Größe des damaligen Teams zurückzuführen, sondern auch auf unzureichende Tests[14]. Die durchgeführten Tests hatten diese Möglichkeit nicht in Betracht gezogen.

Im Kontext der Langzeitarchivierung müssen die aktualisierten Emulatoren anhand geeigneter Tests untersucht werden, bevor sie in das Archiv-Verwaltungssystem aufgenommen werden. Meistens sind diese Tests ausführlich und werden manuell durchgeführt, woraus sich eine Reihe von Problemen ergeben: hoher Testaufwand, hohe Ressource-Kosten, hohe finanzielle Kosten und wiederholte Testfälle. Als Beispiel können die Betreiber von Software - oder musealen - Archivs betrachtet werden. Diese Institutionen haben mit sehr großen Datenmengen an veralteten digitalen Objekten aus unterschiedlichen Betriebssystemen, Anwendungen oder Hardware zu tun.

[10] Der US-amerikanische Medienkonzern ist die Welt größte Unterhaltung-Gruppe, http://www.disney.com

[11] Nur bestimmte Plattformen konnten die durchgeführten Tests bestehen. Dabei wurde das *Intel Pentium-Point Bug* [19] aufgedeckt.

[12] Eine Raumsonde der amerikanischen Raumfahrtbehörde NASA, http://mars.jpl.nasa.gov/msp98/lander

[13] Siehe hierzu http://www.spacetoday.net/Summary/2895

[14] Siehe hierzu http://www.bernd-leitenberger.de/mpl.shtml

Für die einzelnen Systeme sind zahlreiche Tests anhand des jeweils aktualisierten Emulators notwendig. Eine manuelle Durchführung der gesamten Tests wäre nicht denkbar, denn sie wäre mit höheren Kosten (Zeit, Geld und Ressourcen) verbunden.

## 1.3 Ziele

In der Absicht die Langzeitarchivierung von digitalen Objekten zu sichern, wird in dieser Arbeit die Emulationsstrategie bzw. werden die Emulatoren in den Vordergrund gestellt. Die Emulatoren stellen einen langfristigen Zugriff auf digitale Objekte sicher, in dem sie die von digitalen Objekten benötigen Ablaufumgebungskomponenten virtuell verfügbar machen.

Das Ziel der vorliegenden Arbeit ist es, ein System zu entwerfen, das die endgültige Aufnahme eines beliebigen Emulators in einer Archiv-Verwaltungssystem bestimmt. Das zu entwickelnde System soll die Erstellung und die automatische Durchführung von Emulatoren-Tests unterstützen. Die Tests sollen möglichst effizient durchgeführt werden. Der einzige praktische Weg hierfür wäre deren Automatisierung. Dem Anwender des Systems wird die Möglichkeit gegeben, die ausgewählten Tests per Knopf-Druck durchführen zu lassen. Nach jedem Testlauf soll das System ein Testprotokoll zur Verfügung stellen.

# 2 Emulatoren in der Langzeitarchivierung

Die langfristige Betrachtung oder Bearbeitung von digitalen Objekten wird durch die Erfüllung bestimmter technischer Voraussetzungen ermöglicht. Diese beziehen sich größtenteils auf die Ablaufumgebung des betroffenen Objekts. Die Ablauf- oder Abspielumgebung besteht aus der Kombination von Hard- und Softwarekomponenten [33; 6]. In der Langzeitarchivierung digitaler Objekte werden die Anforderungen an solche Komponenten immer größer. Zur Bereitstellung der ursprünglichen Komponente werden die Methoden der Emulation verwendet. In der Umsetzung von Emulationsstrategien spielen Emulatoren eine zentrale Rolle.

## 2.1 Emulatoren -Techniken in der Langzeitarchivierung digitaler Objekte

Die primäre Aufgabe des Emulators ist die Herstellung von einer virtuellen Umgebung anhand von einer realen Umgebung. Die virtuelle Umgebung soll also agieren, als ob sie selbst real sei. Ein User, der dann diese virtuelle Umgebung benutzt, soll keinen Unterschied im Verhältnis zur echten Umgebung merken. Wird die ursprüngliche technische Kontext-Anforderung eines digitalen Objekt erfüllt, kann ein zuverlässiger (Authentizität, Integrität) Zugriff auf das Objekt gewährleistet werden.

### 2.1.1 Struktur von Emulatoren

Emulatoren werden in einer realen Software-Landschaft ausgeführt und stellen eine emulierte (virtuelle) Umgebung bereit. Sie stellen eine Art Schnittstelle zwischen der realen und der virtuellen Welt dar. Obwohl die Beschreibung dieser Schnittstelle nicht immer trivial ist, müssen doch I/O-Geräte nach- und abgebildet werden [33]. Der

Sinclair ZX81[1] zeigte, dass diese Abbildung nicht immer erfolgreich sein konnte, denn dieser Emulator kannte keine heutige PC-Tastatur. Eine Lösung solcher Probleme ist notwendig, um den Emulator in einer aktuellen Umgebung ausführen zu können.
Die Aufstellung der benötigen Komponenten zur Erstellung der Kontext-Umgebung ist keine triviale Aufgabe. Nun wird ein Überblick über zwei Konzepte gegeben, die dabei eine entscheidende Rolle spielen.

### 2.1.2 *View-Path*

Für ein digitales Objekt wird eine Abspielumgebung benötigt, um die Bearbeitung oder Betrachtung des Objekts zu sichern. Diese Abspielumgebung besteht aus Kombinationen von Soft- und Hardware-Komponenten. Eine solche Kombination wird *View-Path* genannt. *A view path is a virtual line of action starting from the file format of a digital object and linking this information to a description of required software and hardware* [7]. Das ursprünglich von der IBM entworfene Konzept wurde inzwischen an der Alberts-Ludwigs-Universität Freiburg im Rahmen des europäischen PLANETS[2]-Projekts verfeinert[3]. Ein aktuelleres Ergebnis von der Universität Freiburg kann in [23] entnommen werden. Mit Hilfe von *View-Path* lässt sich ein digitales Objekt und seine Ablaufumgebung einfacher beschreiben.

### 2.1.3 Softwarearchiv

Das Interesse an Emulation ist in den letzten Jahren stark gestiegen [7]. Es soll ein authentifizierter Zugriff auf digitale Objekte aller Art gesichert werden. Allerdings lösen die Emulatoren nur einen Teil des Problems, denn zusätzliche Software wie Betriebssysteme und Applikationen werden auch dabei benötigt [10]. Für ein gegebenes digitales Objekt enthält das Softwarearchiv die notwendigen Komponenten zur Erstellung der Ablaufumgebung. Welche Komponenten dabei benötigt werden, werden vom *View-Path* festgelegt. Zurzeit gibt es kein etabliertes Verfahren zur Erhaltung von Software. Im Rahmen des PLANETS-Projekts hat sich die Alberts-Ludwigs-Universität Freiburg über den Aufbau und der Verwaltung von einem Softwarearchiv beschäftigt. Die aktuellen Ergebnisse dieses Projekt können in [31; 23] entnommen werden.

[1] Homepage http://www.zx81.de
[2] Preservation and Long-Term Access Through Networked Services. Homepage www.planets-project.eu
[3] Siehe hierzu http://www.ks.uni-freiburg.de/team/dirk/planets-funkt-lza.pdf

## 2.2 Emulatoren als Langzeitzugriff auf digitale Objekte

Ein digitales Objekt ist eine relativ komplexe Menge von Informationen, die in einem bestimmten Format gespeichert werden. Ein solches Objekt ist meist mit einem Informatikprogramm oder einer Anwendung eng verbunden, die verschiedene Handhabungsfunktionalitäten wie die Suche, die Verarbeitung, das Navigieren und die Darstellung anbietet. Zum Beispiel ein *.txt*-Dokument, sprich ein digitales Objekt, kann mit Hilfe eines Editors wie *notepad*[4] oder *gedit*[5] oder auch *Open Office*[6] manipuliert oder betrachtet werden. Es ist also ersichtlich, dass ein digitales Objekt an sich ohne seine technische Umgebung kaum brauchbar wäre. Kritischer wird es dann, wenn diese technische Umgebung auf einem jetzt obsoleten oder unbrauchbaren System läuft. Ausgeführt in einer aktuellen Umgebung ermöglichen die Emulatoren die Rekonstruktion des ursprünglichen Systems.
*A emulator is a device, computer program, or a system that accepts the sames inputs and produces the same outputs as a given system* [29]. Emulatoren sind in der Lage alte Komponenten wie Hardwareplattformen, Betriebssysteme und Applikationen virtuell zu rekonstruieren, besser gesagt diese Komponenten durch eine Software mit identischen Funktionen zu ersetzen, so dass es aus technischer Sicht keinen Unterschied mit den tatsächlichen (echten) Komponenten gibt [31]. So lässt sich beispielsweise eine veraltete x86-Architektur in einer aktuellen Umgebung (Ubuntu 10.04 LTS herausgegeben im April 2010) mit Hilfe der freien virtuellen Maschine QEMU emulieren. Ein Abbild davon zeigt die Figur 2.1. Emulatoren machen es also möglich eine alte Umgebung in einer aktuelleren Umgebung auszuführen. Sie stellen eigentlich eine Art Brücke zwischen der Vergangenheit und der Gegenwart dar. Für die zu emulierenden Systeme werden die Eingabe-Parameter benötigt, um die erwarteten Ausgaben zu produzieren. Deshalb müssen sich Emulatoren um die geeignete Umsetzung der Ein- und Ausgabesteuerung bemühen [31, S. 81].

### 2.2.1 Ansatzpunkte

Im Kontext der Langzeitarchivierung digitaler Objekte sind folgende Punkte für den Einsatz von Emulatoren von Relevanz [33]:

[4] Ein einfacher Text-Editor für Betriebssystem Windows. Es wurde seit Windows 1.0 in Microsoft Windows eingeführt, http://www.notepad.org
[5] Der offizielle Text-Editor für GNOME-Desktop, http://projects.gnome.org/gedit
[6] Ein freies Office-Paket, http://www.openoffice.org

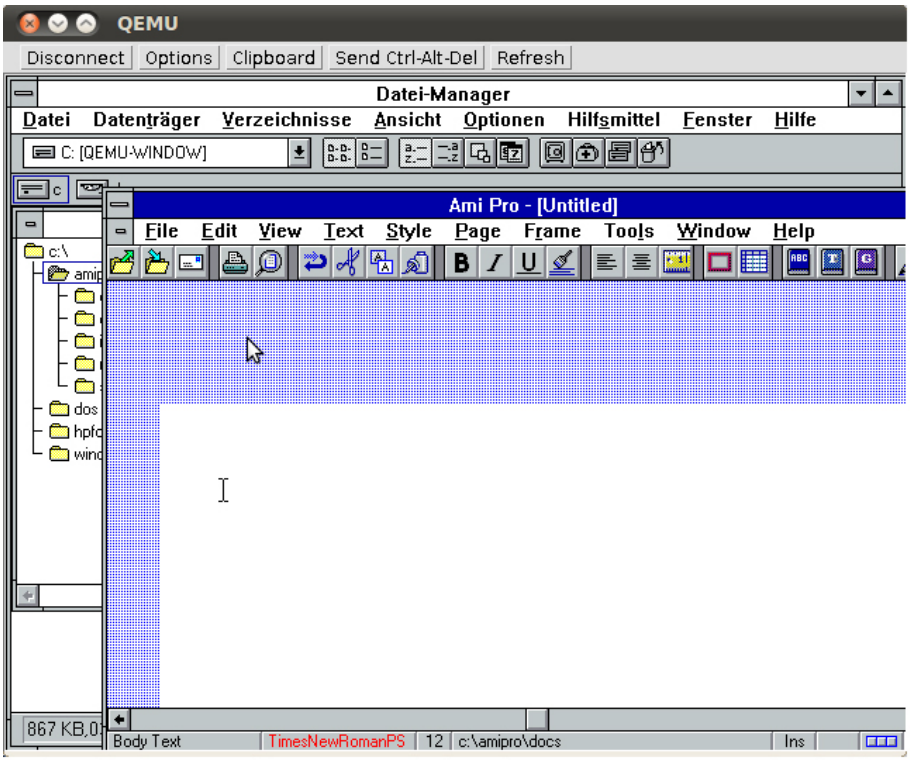


Abbildung 2.1: Windows 3.11 unter Ubuntu 10.4

### Hardware-Emulation

Die Aufgabe von Emulatoren-Strategien besteht darin, eine Hardware oder Rechnerarchitektur in Software nachzubilden. Ein eklatantes Beispiel hierfür stellt die freie virtuelle Maschine QEMU dar. QEMU schafft eine reale Abkopplung der emulierten Systeme (Gast-Maschine) mit den darunter liegenden Hardware-Ressourcen der Host-Maschine.

### Betriebssystem-Emulation

Hier geht es um die funktionale Nachbildung einer Schnittstelle und die der Bibliotheken eines Betriebssystems. Es ist also möglich Anwendungen in einem fremden Betriebssystem auszuführen. Ein Beispiel hierfür wäre der Terminal Cygwin[7], der es erlaubt Linux-Api unter Windows auszuführen und gleichzeitig wird auch ermöglicht

[7] Get that Linux feeling - on Windows! Homepage: http://www.cygwin.com/

unter Cygwin auf das darunter liegende Host-System zuzugreifen. Der weit verbreite Emulator Wine[8] wäre auch ein weiterer Vertreter dieser Emulation-Art.

#### Applikations-Emulation

Dadurch wird ermöglicht, alte Datenformate wieder lesbar zu machen.

Wenn die Emulatoren in der Langzeitarchivierung verwendet werden sollen, werden sie meistens als Hardware-Emulator eingesetzt, denn diese Strategie ermöglicht eine Abstraktion der Hardware-Schicht [31]. Somit hängt die emulierte Umgebung nicht von der darunter liegenden Host-Maschine ab. Für die neue entwickelte Hardware sollen lediglich die Emulatoren angepasst (z.B.: per Update) werden (s. Abbildung 1.1).

### 2.2.2 Rolle

Nachfolgend soll die Rolle bzw. Aufgabe eines Emulators zusammengefasst werden. Ein Emulator:

- bewahrt die Integrität und die Authentizität beim Zugriff auf ein digitales Objekt in der emulierten Umgebung.
- bleibt unabhängig (abgesehen vom Betriebssystem) der Hardware- und Softwareumgebung.
- beweist, dass Emulation eine lebensfähige Strategie in der digitalen Langzeitarchivierung ist, denn er sichert die Erhaltung der ursprünglichen Umgebung, in der das digitale Objekt erzeugt wurde.

### 2.2.3 Langzeitarchivierung von Emulatoren

Emulatoren sind eigentlich Software, daher benötigen sie auch eine passende Kontext-Umgebung um ausgeführt zu werdenb. Eine dauerhafte Ausführung von Emulatoren ist nur möglich, wenn diese an die aktuelle Umgebung angepasst werden. Darum bemühen sie sich die Emulatoren-Hersteller. Es ist auch von großem Vorteil, wenn die Software reichlich dokumentiert wird. Dies vereinfacht ihre Anpassung an andere (aktuelle) Hard- und Software-Landschaften. In [30] werden ein paar Möglichkeiten zur dauerhaften Aufbewahrung von Emulatoren vorgestellt.

[8] **W**ine **i**s **n**ot a **e**mulator. Homepage: http://www.winehq.org/

# 3 Emulatoren-Testing im Kontext der Langzeitarchivierung

Nach dem im letzten Kapitel gegebenen Überblick über Emulatoren im Kontext der Langzeitarchivierung soll hier ein Konzept für den Test von Emulatoren im Kontext der Langzeitarchivierung entwickelt werden. Im nächsten Kapitel wird die Implementierung des Konzepts genauer beschrieben.

Da Emulatoren eigentlich Software sind, wird zuerst ein Überblick über das Software-Testing im Allgemeinen gegeben.

## 3.1 Software-Testing

Aus dem IEEE[1] 829[2] im ISQTB[3] GLOSSAR veröffentlichten Bericht wird ein Test als eine Menge von einem oder mehreren Testfällen definiert.
*Software testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item* [29].
Eine Software wird nach bestimmten Spezifikationen implementiert und sie soll einen Algorithmus konkret darstellen [17]. Während der Test-Phase soll die Software nach diesen Spezifikationen untersucht werden. Die Software-Testing-Phase umfasst generell zwei Aspekte:

[1] IEEE: *Institute of Electrical and Electronic Engineers* ist ein internationaler Verband aus Elektrotechniker und Informatiker. Der Verband bildet ein Gremium und ist der Autor von mehreren Standardisierungen im Technik-, Software- und Hardwarebereich. http://www.ieee.org

[2] IEEE 829 bezeichnet der Standard für die Test Dokumention. Dies wurde vom IEEE initiiert.

[3] Das ISQTB (*International Softaware Testing Qualifications Board*) beschäftigt sich mit der Standardisierung der Ausbildung für professionelle Softwaretester. Es wird u.a. unterschiedlichen Titel verliehen wie das *ISTQB Certifier Tester*, das ist eine Bezeichnung für eine standardisierte Qualifikation zum Softwaretester. Homepage http://www.istqb.org

- Qualitätsnachweis: Die Software soll sich konform zu den Anforderungen verhalten.
- Fehlerfindung: vor der Auslieferung sollen mögliche Fehler gefunden werden.

### 3.1.1 Test-Schritte

Zuerst wird ein Überblick über die verschiedenen Schritte gegeben, die während eines Software-Testing in Betracht kommen. Diese Schritte sind vom ISTQB und von anderen Institutionen vorgegeben. Diese einzelnen Aktivitäten müssen nicht nacheinander ausgeführt werden:

#### Testplanung und Steuerung

*A test plan is a document describing the scope, approach, resources, and schedule of intended test activities. It identifies test items, the features to be tested,the testing tasks, who will do each task, and any risks requiring contingency plans*[29]. Die Planung wird im Allgemeinen am Anfang der Softwareentwicklung durchgeführt, aber diese kann auch durchaus im Laufe der Entwicklungsphase durchgeführt werden.

#### Testanalyse und Testdesign

Die Testplanung gibt die Richtung vor, die während der Testphase befolgt werden soll. Um die Test-Durchführung zu erleichtern, soll das Soll-Verhalten der zum testende Software klar sein. In diesem Test-Schritt wird dann festgelegt, wie die im Testplan aufgeführten Tests aussehen und ablaufen müssen.

#### Testrealisierung und Testdurchführung

Hier findet die eigentliche Test-Ausführung statt. Diese kann also manuell, teilautomatisiert oder automatisiert erfolgen.

#### Testauswertung und Bericht

Die in der Testdurchführung erzielten Ergebnisse werden in diesem Test-Schritt ausgewertet und anschließend protokolliert.

## 3.2 Prinzipien des Emulatoren-Testing

Unabhängig vom verwendeten Emulator sind Kenntnisse über die technische Kontext-Umgebung des digitalen Objekts notwendig [7]. Zur Entwicklung des Konzepts wird angenommen, dass für jedes digitale Objekt die Informationen (*View-Path* und Softwarearchiv) zur Bereitstellung der Ablaufumgebung bereits vorliegen. Die Bereitstellung und die Verwaltung der Ablaufumgebung eines gegebenen digitalen Objekts sind nicht Teil dieser Arbeit. Allerdings können die Details hierfür aus [23] entnommen werden. Eigentlich basiert das hier entwickelte System zum Teil auf den Erkenntnissen aus [23] und [25].
Zur Entwicklung des Frameworks sollen einige Test-Strategien vorgestellt werden und zugleich wird überprüft, wie die jeweilige Strategie zur Nutzung des Konzepts eingesetzt werden könnte.

### 3.2.1 *White-Box-Testing*

Die aktualisierten Emulatoren-Softwares sollen anhand geeigneter Tests untersucht werden. Dabei soll u.a geprüft werden, ob die Softwares nach ihren intern beschriebenen Algorithmen fehlerfrei ausführbar sind. Das *White-Box-Testing* unterstützt drei Typen von Testing:

1. *Unit Testing*: das entspricht dem Test einer Software Komponente, die nicht mehr in weitere Komponente unterteilt werden kann. Dies ist von Bedeutung, wenn eine Komponente in eine andere integriert werden soll. Somit wird die Anzahl der Fehler auf dem Endprodukt deutlich reduziert.

2. *Integration Testing* untersucht die Kombination von Hardware Komponenten oder Software Komponenten miteinander.

3. *Regression Testing*: Hier geht es darum ein System oder eine Komponente wiederholt zu testen, um zu überprüfen, ob die an den Systemen oder Komponenten vorgenommenen Änderungen keine Funktionsstörung verursacht haben.

Die Anwendung des *White-Box-Testing* erfordert vom Anwender gewisse Kenntnisse über die interne Struktur des ausgewählten Emulators sowie Programmierungskenntnisse, um die Testfälle zu entwerfen und auszuführen. Solche Kenntnisse sind für das hier entwickelte Framework nicht von Relevanz. Es lohnt sich an dieser Stelle die Aufgabe des Emulatoren-Testing-Frameworks wieder in Erinnerung zu rufen: Das Framework soll anhand geeigneter Tests überprüfen bzw. testen, ob die bisherigen

Funktionalitäten eines Emulators in seiner aktualisierten Version erhalten geblieben sind. Das Framework erhält als Eingabe eine Testfall-Liste und einen gegebenen Emulator. Nach der Durchführung des Tests soll ein Testprotokoll vorliegen (s. Abbildung 3.1), das die Ergebnisse der Tests enthält. Dabei werden die externen Funktionalitäten des Emulators unter die Lupe genommen. Daher wird das *Black-Box-Testing* für


Abbildung 3.1: Emulatoren-Testing-Framework

das Emulatoren-Testing-Framework in Betracht gezogen.

### 3.2.2 Black-Box-Testing versus Emulatoren-Testing

Das primäre Ziel des *Black-Box-Testing* ist es, anhand von Tests herauszufinden, ob eine Software nach ihren externen funktionalen Anforderungen reibungslos ausführbar ist. Es ist dabei als wichtig zu betonen, dass die erfolgreiche Durchführung dieser Tests nicht garantiert, dass die Software ein fehlerfreies Produkt ist. Im *Black-Box-Testing* werden hauptsächlich folgende Software-Eigenschaften getestet [34]:

1. Fehlerhafte Funktionalität
2. Interface-Fehler
3. Fehler in der Datenstruktur
4. Verhalten- oder Effizienzfehler
5. Initialisierungs- und Terminierungsfehler

Das im Emulator-Testing-Framework angewandte Testprinzip wird zum größten Teil vom *Black-Box*-Testprinzip abgeleitet.
Zur Anwendung der *Black-Box*-Testing-Technik sind weder die Kenntnisse über die internen Strukturen der Software noch die Programmierungskenntnisse zum Design

der Testfälle erforderlich. In [34] wird sogar empfohlen, dass der Softwaretester nicht der Programmierer sein sollte. Es geht hier darum, den Emulator auf bestimmte Funktionalitäten zu testen. Für gegebene Eingabedaten (*Input-Data*) soll vom Framework das Verhalten (*Ouptput-Data*) des Emulators geprüft werden. Somit wird für den Anwender des Frameworks die innere Struktur des Emulators abstrahiert. Die Abbildung 3.2 stellt einen ersten Eindruck dieser Technik dar. Der ausgewählte

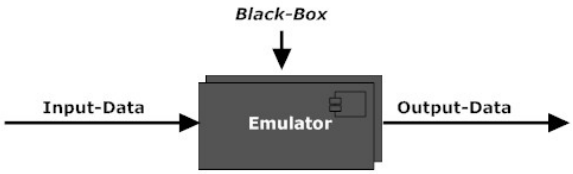


Abbildung 3.2: Emulator als Black-Box

Emulator wird als eine Art *Black-Box* betrachtet. Der Code und die interne Struktur des Emulators sind hier überflüssig. Wie aus Abbildung 3.2 zu entnehmen ist, bekommt der Emulator *Input-Data* und erzeugt dafür die passende *Output-Data*. Der Softwaretester hat lediglich Kenntnisse der Input-Daten. Wichtig beim *Black-Box-Testing* oder funktionalen Testing sind die vom Emulator generierten Ausgaben (*Output-Data*). Diese Ausgaben werden vom Framework verwendet, um den Testschritt auszuwerten. Sollten die Ergebnisse der Auswertung nicht den Erwartungen entsprechen, dann wird vom System eine Fehlerbehandlungsprozedur gestartet. Mehr dazu im Kapitel 3.3.2.

Um die Verlässlichkeit der Emulatoren sicherzustellen, sollen die aktualisierten Emulatoren auf die bisherigen Emulatoren-Funktionalitäten untersucht werden. Hierzu wird das *Regression Testing* angewandt. Im weiteren Verlauf der Arbeit wird dieser Test-Typ im Zusammenhang mit dem Emulatoren-Testing etwas ausführlicher vorgestellt.

### 3.2.3 Regression Testing

*Regression Testing is a selective retestting of a system or a component to verify that modifications have not caused unintented effects and that the system or component still complies with its specified requirements* [29]. Die erzeugten Testfälle sollen wiederholt auf den aktualisierten Emulatoren durchgeführt werden. Durch den Ansatz eines *Regression Testing* soll eigentlich geprüft oder getestet werden, ob der aktualisierte Emulator (*Black-Box*) die bisherigen Funktionen erfüllt. Es soll auch betont werden, dass das Ziel nicht jenes ist, alle möglichen Tests durchzuführen, denn es

geht nicht darum, den Emulator durchzutesten, sondern darum, den aktualisierten Emulator durch geeignete Tests auf bestimmte Funktionalitäten zu untersuchen. Diese Funktionalitäten sind mehr oder weniger auf den früheren Emulatoren-Versionen vorhanden.
Die vorgenommenen Änderungen auf die aktualisierten Emulatoren sollen nicht die bisherigen Funktionalitäten des Emulator beeinträchtigen. Diese Aussage kann nur mit Hilfe von Tests bestätigt werden. Im Falle eines Archivs werden mehrere Systeme involviert und jedem einzelnen System werden eigene Tests bereitgestellt, so dass eine manuelle Durchführung der gesamten Tests mit hohen Kosten (Zeit, Geld, Ressourcen) verbunden ist. Deshalb wird das hier entwickelte Framework die Testautomatisierung unterstützen.

## 3.3 Testautomatisierung

Die primären Ziele einer automatisierten Test-Durchführung bestehen einerseits darin, die Anzahl der manuellen Aktivitäten zu minimieren, anderseits die Durchführung von möglich vielen Testfällen innerhalb kurzer Zeit zu ermöglichen. Es soll ein Weg gefunden werden, um die zahlreichen durchzuführenden Tests effizienter laufen zu lassen. *Eine Testautomatisierung ist ein Ansatz von Softwarewerkzeugen zur Erstellung bzw. Programmierung von Testfällen mit dem Ziel, die Testfälle rechnergestützt wiederholt auszuführen zu können* [4]. Es wird möglich sein, eine Metrik der Anzahl der erfolgreichen Tests nach jedem Testlauf zu liefern.
Der aktualisierte Emulator soll auf bestimmte Funktionen getestet werden. Diese Funktionen sind u.a.:

- Betriebssystem aus verschiedenen Medien (Festplatte, CD-ROM oder Floppy) booten
- Netzwerk überprüfen
- Eine Anwendung starten lassen und bestimmte Aktionen ausführen
- Eine Anwendung installieren
- Tastatureingabe prüfen

Es wird festgestellt, dass diese Funktionen sich am besten per Benutzereingabe ausführen lassen. Es wird eine Interaktion zwischen der emulierten Umgebung und dem Anwender geben und diese wird auf einer Benutzeroberfläche stattfinden. Außerdem soll erreicht werden, dass diese Aktionen automatisch durchgeführt werden können, um die Testkosten zu sparen und die Testauswertung zu vereinfachen. Hierzu

wird auf ein Netzwerk-Protokoll zurückgegriffen, welches erlaubt, Benutzereingaben (Maus- und Tastatureingabe) zu steuern und auf den Bildschirminhalt zuzugreifen. Das sicherlich passendste Protokoll dafür wäre das RFB-Protokoll [24], das von einer systemunabhängigen VNC-Verbindung unterstützt wird. Somit wäre ein gutes Werkzeug für die Tests gefunden.

### 3.3.1 Testwerkzeug: VNC

Für das Testumfeld gibt es eine Host- und eine Gastmaschine. Die Hostmaschine ist die physikalische Maschine, wo der Archivverwalter sitzt. Die Gastmaschine dagegen ist die virtuelle Umgebung, die von einem beliebigen Emulator bereitgestellt wird. Der Emulator an sich selbst ist eine Software, die in der Hostmaschine ausgeführt wird. Von der Hostmaschine aus soll es möglich sein, die Benutzereingabe zur virtuellen Umgebung (Gastmaschine) weiterzuleiten. Die Informationen über den Bildschirminhalt der Gastmaschine sollen für die Hostmaschine zugänglich sein. Diese gesamten Datentransfers werden vom RFB-Protokoll ermöglicht, das selbst von einer VNC-Verbindung unterstützt wird. Die jeweiligen Maschinen (Host- und Gastmaschine) sollen daher ein VNC ausführen bzw. sie sollen eine VNC-Schnittstelle zur Verfügung stellen. Das RFB Protokoll verleiht dem VNC eine hohe Flexibilität, es lohnt sich also einen Blick darauf zu werfen.

#### RFB-Protokoll

RFB steht für Remote Frame Buffer, ein einfaches Netzwerkprotokoll, das den Fernzugriff auf die graphische Oberfläche einer anderen Maschine erlaubt. Entwickelt wurde das Protokoll vom Olivetti Research Laboratory (ORL) und das RFB diente zum Remote Display für ein Thin Client. Nach der Entwicklung vom VNC wurde die Verwendung des RFB Protokolls noch stärker geprägt. Heute wird das Protokoll von der Firma RealVNC Ltd weiterentwickelt. Die aktuellste RFB Version `3.8` wird im Emulatoren-Testing-Framework eingesetzt. Das Protokoll RFB ist systemunabhängig, denn es wird auf der Framebuffer-Ebene ausgeführt. Bei der Ausführung des RFB Protokolls kommen grundsätzlich zwei Akteure zum Ausdruck:
**RFB Client**: das ist der Remote Endpunkt, wo der Benutzer sitzt. Hier werden die Tastatur und Mauseingabe initiiert.
**RFB Server**: wird vom Endpunkt dargestellt, wo der Framebuffer initiiert werden.

Der Bildschirminhalt des Servers ist die zentrale Nachricht, die dem Client versendet wird. Die Abbildung 3.3 zeigt den Ausschnitt der Verwendung des RFB Protokolls. Der Aufbau der Verbindung zwischen Client und Server verläuft in drei Phasen:

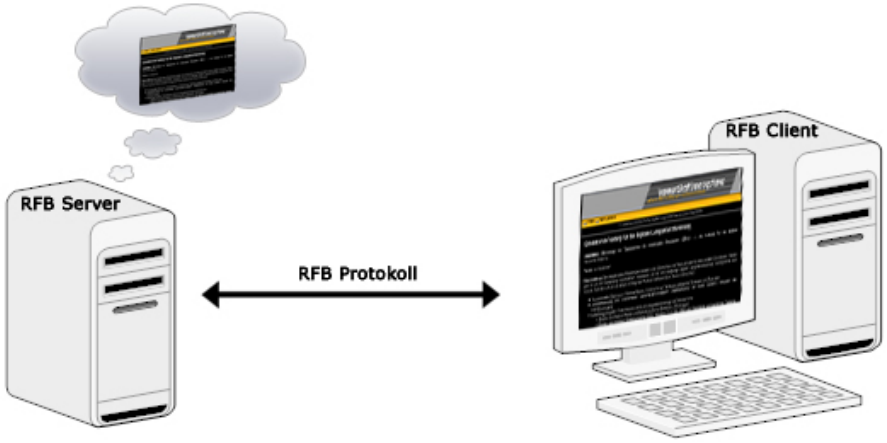


Abbildung 3.3: Ausschnitt eines RFB Protokolls

1. *Handshaking*: das ist der erste Schritt in der Aushandlung der Verbindung. Die RFB Protokoll-Version und die Sicherheitsregeln werden ausgehandelt.

2. Initialisierung: Es wird zwischen Client und Server festgelegt, welches Format (z.B.: 16-Bit oder 24-Bit *true color*) und welche Kodierung für das Versenden der Pixel Dateien verwendet wird. Die vom Server bereitgestellten Pixel Dateien müssen vom Client unterstützt werden.

3. Ausführung: Die Details der Verbindung wurden bereits festgelegt. Nun kann die eigentliche Protokoll-Interaktion stattfinden.

Wenn die eigentliche Verbindung zwischen Client und Server besteht, werden dabei Nachrichten ausgetauscht. Diese sind von zwei Typen:
**Framebuffer update**: Diese Nachricht wird per Anfrage vom Server dem Client geschickt, wenn der Framebuffer von einem gültigen Stand zu einem anderen übergeht. Es werden dabei primitive Nachrichten verschickt, der Art: *"put a rectangle of pixel data at a given x,y position"*
**Input protocol**: Sollte die Hostmaschine irgendeine Art von Ereignissen (Maus oder Tastatur) initiieren, wird das entsprechende Event der Gastmaschine geschickt.
Nähere Informationen über die Spezifikationen des hier verwendeten RFB Protokolls können im [24] entnommen werden.

Aus der Basis des RFB Protokolls wird die VNC-Verbindung aufgebaut, die ebenfalls aus einem VNC Server und Client besteht. Der VNC Server unterstützt den RFB Server und genauso unterstützt der VNC Client den RFB Client. Also werden der

VNC Server auf das Gastsystem und der VNC Client auf das Hostsystem ausgeführt. Das Gastsystem wird vom Emulator dargestellt und wie schon erwähnt wurde, wird der Emulator wie eine *Black-Box* betrachtet. Es wird kein Zugriff auf die interne Struktur und auf den Kode des Emulators gewährt. Daher sollen geeignete Emulatoren ausgewählt werden. Bevor die Auswahlkrtiterien des Emulators detailliert vorgestellt werden, soll erstmals die Aufgabe des VNC Client im Emulatoren-Testing noch klarer werden.

### 3.3.2 VNC Client im Emulatoren-Testing

Der VNC Client wird auf die Hostmaschine ausgeführt und dies ermöglicht, die Gastmaschine bzw. die emulierte Umgebung zu steuern. Das darunter liegende RFB Protokoll ermöglicht den Zugriff auf den Framebuffer der Gastmaschine und die Steuerung ihrer Tastatur und Maus. Zum Einsatz im Framework wird die im Rahmen des PLANETS-Projekts vorgestellte GRATE[4]-Architektur verwendet. Die Abbildung 3.4 zeigt einen Ausschnitt dieser Architektur. Die GRATE-Architektur hat eine Abstraktionsschicht zwischen der Host- und Gastmaschine. Diese Abstraktionsschicht wird von einer VNC-Schnittstelle bereitgestellt. Löst ein Benutzer aus der Hostmaschine ein beliebiges Event (Maus- oder Tastaturevent) aus, wird dieses in der virtuellen Umgebung (Gastmaschine) dementsprechend per Emulation reproduziert. Mit Hilfe des RFB Protokolls bleibt der Zugriff auf den Bildschirminhalt der Gastmaschine weiterhin erhalten. Eigentlich findet keine direkte Verbindung zwischen Host- und Gastmaschine statt. Die VNC-Schnittstelle übernimmt die ganze Verbindungsabwicklung. Durch den Einsatz der GRATE-Architektur sind die emulierte und reale Umgebung voneinander unabhängig. Für eine beliebige Hostmaschine könnte eine beliebige Gastmaschine bzw. ein beliebiger Emulator eingesetzt werden. Eine notwendige Voraussetzung dafür wäre von jeweiliger Umgebung, eine VNC-Schnittstelle bereitzustellen. Da das RFB Protokoll auf einer TCP/IP Verbindung aufgebaut wird, würde der Zugriff auf die emulierte Umgebung nicht Computer-gebunden sein und somit könnte die eingesetzte GRATE-Architektur auf ein breiteres Netzwerk wie Internet ausgeführt werden. Dies wäre für das Emulator-Testing-Framework schon ein günstiger Ausgangspunkt.

Das Emulator-Testing-Framework wird via Internet bereitgestellt. Die darunter liegenden technischen Details (z.B.:Softwarearchiv und *View Path*, Bereitstellung von Emulatoren) werden vom Archivverwalter geregelt und sind für den Anwender abstrahiert. Vom Anwender werden Kenntnisse über die alten emulierten Umgebungen benötigt, um überhaupt die Testfälle erzeugen zu können, denn diese bestehen aus

[4]Global Remote Access To Emulation

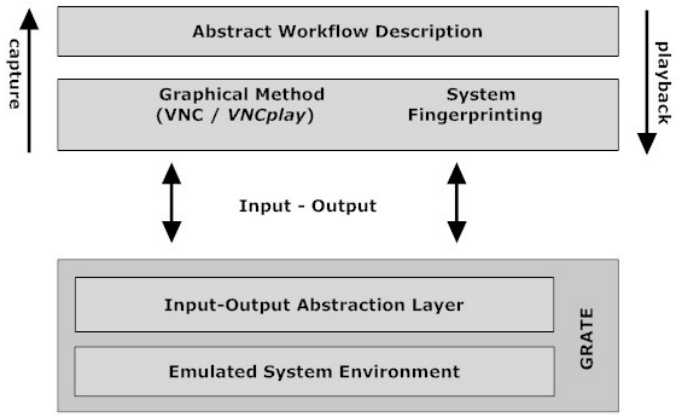


Abbildung 3.4: GRATE-Architektur

Benutzereingaben. Diese Remote-Strategie bringt mehrere Vorteile mit sich wie bereits in einem vergleichbaren Bericht [7] festgestellt wurde. U.a.:

- Hohe Flexibilität: der Service wird über Internet zugreifbar sein.
- Das Service-Management erfolgt auf einer einzigen Maschine (die Hostmaschine); u.a. neue Software und neue Betriebssysteme ins Archiv hinzufügen, neue Emulatoren bereitstellen.
- Kein Problem mit den Lizenzen, denn die Software werden nicht auf die private Maschine der Benutzer kopiert.
- Zur Ausführung des Emulatoring-Tests wird eine minimale System-Anforderung benötigt.

#### Auswahlkriterien geeigneter VNC Client

Die Testfälle sollen automatisch durchführbar sein und bestehen hauptsächlich aus Benutzereingaben. Also soll der ausgewählte VNC Client die automatische Durchführung von Benutzereingaben ermöglichen. Bevor die Testfälle sprich Benutzereingaben automatisch durchgeführt werden, muss eine vorherige Aufnahme dieser Benutzereingaben stattgefunden haben. In einem ersten Durchgang (Testfall-Erstellung oder Aufnahmephase) muss der VNC Client so gestartet werden, dass die vom Benutzer initiierten Eingaben protokolliert werden. In einem zweiten Durchgang (Testfall-Durchführung oder Wiedergabephase) werden die im Protokoll gespeicherten Benutzereingaben wieder eingelesen. Außerdem soll es möglich sein, die gespeicherten Benutzereingaben so oft wie benötigt in unterschiedlichen Systemen abzuspielen.

### Geeignete Remote Desktop für den Emulatoren-Testing: VNCplay

Im Jahr 2005 wurde an der Standford's Universität ein Java-basierter VNC Client mit dem Name *VNCplay* [21](eine modifizierte Version des Java VNC Client TightVNC[5]) entwickelt. Der *VNCplay* ist ein plattformübergreifendes Tool zum Messen Interaktiven VNC Session auf GUI-basierten Systemen [21]. Der *VNCplay* ist in der Lage eine interaktive Benutzer-VNC-Session aufzunehmen und diese in verschiedenen Systemen mit unterschiedlichen Einstellungen wieder abzuspielen.

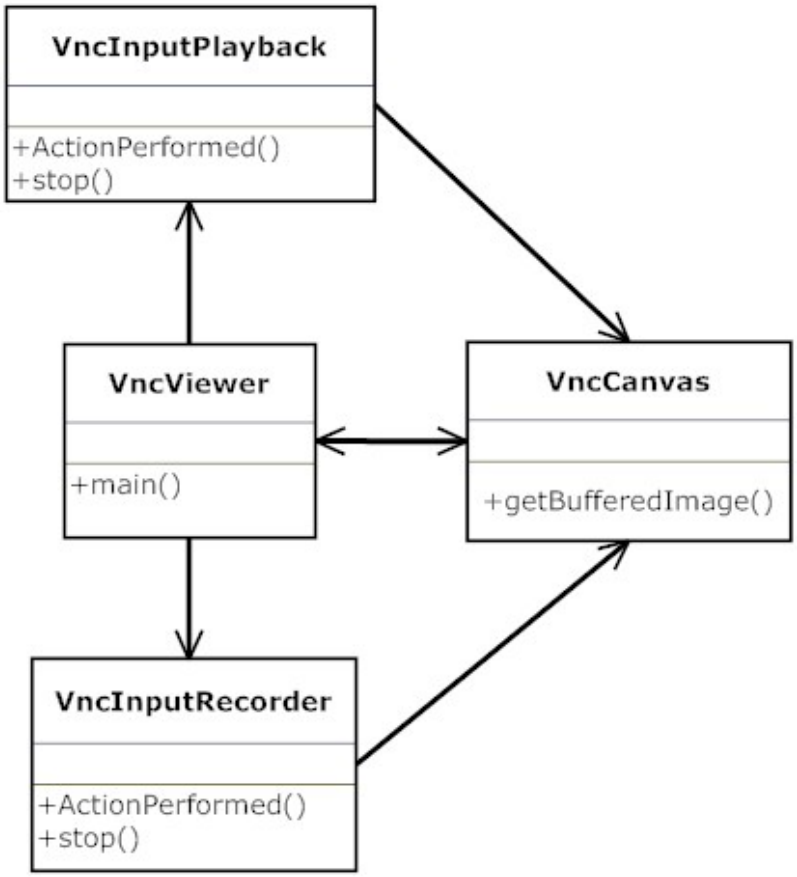


Abbildung 3.5: Ausschnitt aus dem Klassendiagramm vom *VNCplay* - *Quelle[25]*

[5]TightVNC web page. http://www.tightvnc.com

### How it's works?

Der *VNCplay* besteht aus drei Komponenten: dem *Recorder*, dem *Replayer* und dem *Analyser*. Der *Analyser* wird in dieser Arbeit nicht betrachtet. Die Abbildung 3.5 zeigt einen Ausschnitt des Klassendiagramms des *VNCplay*. Die Klasse *VncInputPlayback* bzw. *VncInputRecorder* ist für die Wiedergabe bzw. die Aufnahme zuständig.
Die Abbildung 3.7 stellt die Wiedergabe- und Aufnahmephase beim *VNCplay* dar. Der in der Hostmaschine (*Host-System*) ausgeführte VNC Client (*VNCplay*) baut eine Verbindung mit dem im Emulator ausgeführten VNC Server auf. Durch diese VNC-Schnittstelle werden die Daten zwischen Host- und Gastmaschine (*Guest-System*) ausgetauscht. In der Aufnahmephase werden die Benutzereingaben vom *Host-System* initiiert. Diese werden vom *VNCplay* abgefangen und nacheinander als Zeichenkette in der *Log*-Datei (s. 3.6) gespeichert. Fängt der *VNCplay* ein *KlickEvent* ab, dann schickt er dem VNC Server eine *getBufferedImage()*-Nachricht. In der emulierten Umgebung wird vom VNC Server um den Mauszeiger herum ein quadratisches Bildschirmfoto (`46x46`) gemacht und dem *VNCplay* zurückgeschickt. Aus diesen Informationen wird der *Sync-Point* berechnet und in der *Log*-Datei (s. 3.6) gespeichert. Das Bildschirmfoto `46x46` reduziert die Größe der *Log*-Datei und vermeidet die Betrachtung von irrelevanten Bildschirmpunkten (z.B.: die Systemuhr). Nach diesem Vorgang kann der eigentliche *KlickEvent* in der emulierten Umgebung ausgeführt werden.
In der Wiedergabephase werden die in der *Log*-Datei gespeicherten Informationen vom *VNCplay* wieder ausgelesen, um die exakten Interaktionen in der emulierten Umgebung genauso wie während der Aufnahmephase zu reproduzieren. Allerdings wird die Wiedergabe nicht blind ausgeführt, dabei helfen die gespeicherten *Sync-Point*. Diese Werte werden mit den tatsächlichen Werte aus dem Bildschirm der Gastmaschine verglichen (Pixel-per-Pixel Vergleich). Nur bei Übereinstimmung der Werte kann der entsprechende *KlickEvent* ausgeführt werden. Die in der *log*-Datei gespeicherten Benutzereingaben sind von zwei Typen:

- *MouseEvent*: das sind alle aus der Maus erzeugten Ereignisse wie Klick, Doppelklick, Mausbewegung.

- *KeyEvent*: diese Ereignisse werden von der Tastatur ausgelöst.

Ein Ausschnitt der *log*-Datei wird in der Abbildung 3.6 dargestellt. Die *log*-Datei speichert ebenfalls wo und wann ein Ereignis (*MouseEvent*, *KeyEvent*) stattgefunden hat, damit während der Wiedergabe diese exakte Stelle getroffen wird. Die Wiedergabe würde somit genau so lang wie die Aufnahme dauern, falls keine Beschleunigung stattfindet. Wie aus der Abbildung 3.6 zu entnehmen ist, gehört zu jedem


Abbildung 3.6: Ausschnitt einer *Log*-Datei

gespeicherten Event-Type ein `id`. Somit kann das System im Falle eines *MouseEvent* unterscheiden, ob es sich um ein *MousePressEvent*(`id=501`) oder ein *MouseReleaseEvent*(`id=502`) handelt. Mit den gleichen Prinzipien werden auch die *KeyEvent*s aussortiert. Aufgrund der in der *log*-Datei gespeicherten Informationen ermöglicht VNC, eine aufgenommene Benutzer VNC-Session so oft wie möglich automatisch wiederzugeben.

Das Framework stellt dem Endbenutzer (Softwaretester) ein Web-Interface zur Erstellung und Durchführung von Testfällen zur Verfügung. Das Applet *VNCplay* wird in der Webseite eingebettet. Dadurch hat der Endbenutzer Zugriff auf die emulierte Umgebung. Der Softwaretester arbeitet, als ob er sich auf der Hostmaschine befindet. Die Test-Durchführung und -Erstellung kann also auf einem beliebigen Gerät ausgeführt werden. Im Kapitel 4.3.1 werden die genaueren Voraussetzungen, die dabei erfüllt werden müssen, detailliert vorgestellt.

Allerdings wie es im [21] gezeigt wurde, kann es während der Wiedergabe zu Fehlern kommen. Bei den komplexen Betriebssystemen wie Windows oder Linux kann es zur Verzögerung beim Zeichnen eines Windows-Fenster oder einer Komponente in einem GUI kommen. Dies führt dazu, dass zwar ein Klick am richtigen Punkt getätigt wird, aber der Zeitpunkt nicht passend ist. Sollte ein Klick zum Beispiel auf einem Knopf *"OK"* erfolgen, würde ein frühzeitiger Klick an dieser Stelle nicht zum erwarteten Ergebnis führen. Zur Lösung dieses Problem haben die Entwickler das *"wiggle mouse"*-System eingeführt. Das besteht darin, die Maus ganz langsam um `1px` zu bewegen, bis das Hintergrund-Bild dem erwarteten *Sync-Point* entspricht. Dabei wird u.a. erreicht, dass das darunter liegende Event (z.B.: *onMouseOver*) ausgelöst wird, denn solche Event führen meistens zu einem Highlight des Hintergrund-Bildes.

Zur Behandlung der Fehler, die während einer Wiedergabephase auftreten können,

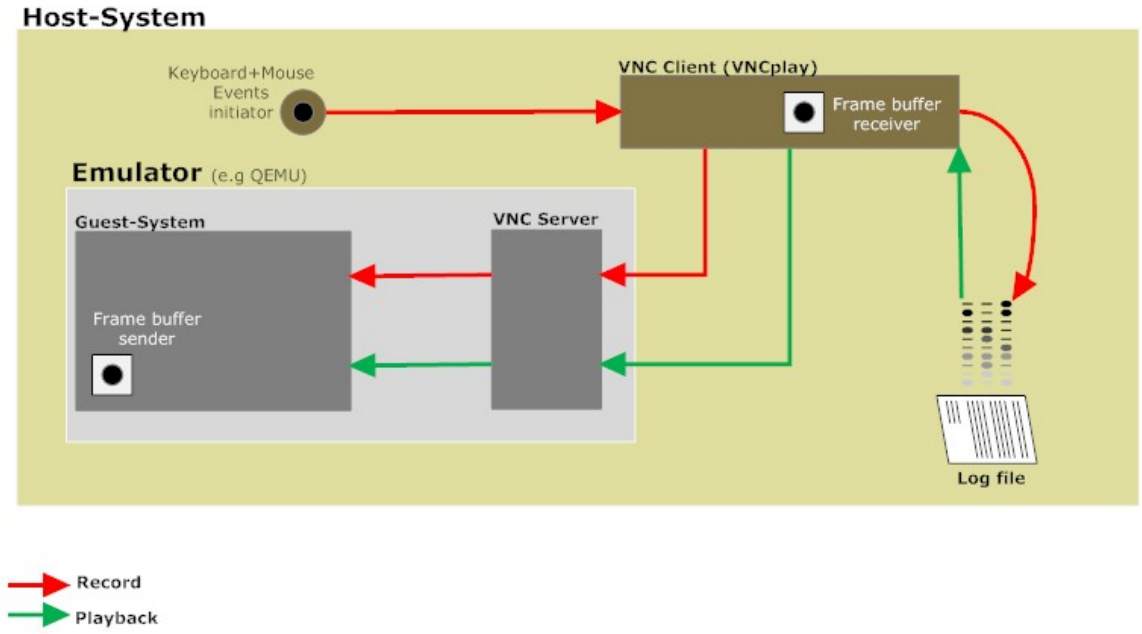


Abbildung 3.7: *VNCplay Record* und *Playback*

wurden an der Alberts-Ludwigs-Universität Freiburg einige Verbesserungen des *VNCplay* vorgestellt:

**1. Verbesserung**:
Zur Erkennung und Behandlung von Fehlern, die während einer Wiedergabe entstehen können, wurde im [25] eine relativ triviale Lösung vorgeschlagen: Die Aufnahme sollen nun Zustand-basiert und nicht mehr Zeit-basiert ablaufen. Ein Zustand dient in diesem Fall als Referenzpunkt während der Wiedergabe und besteht lediglich aus einem Mausklick. Jeder Zustand wird mit der sogenannten *pre-* und *postCondition* verknüpft. Erst wenn die *preCondition* erfüllt wird, kann das bestehende Ereignis (Mausklick) ausgeführt werden. Anschließend wartet das System auf die Erfüllung der *postCondition*, bevor ggf. der nachfolge Zustand eingeschaltet wird. So können Fehler in der Bearbeitung gezielt behandelt werden. Der Vorteil gegenüber einer Zeit-basierten Wiedergabe liegt in der Möglichkeit auf vorhergehende Zustände zu springen.
**2. Verbesserung**:
Zur Einrichtung eines soliden Werkzeugs zur Durchführung der Testfälle wurde in dieser Arbeit eine weitere Verbesserung des ursprünglichen *VNCplay* implementiert. Diese neue Version vereinigt die Erkenntnisse die glanzvoll im [21] und im [25] vorgestellt wurde. Der hier verwendete *VNCplay* ist ebenfalls Zustand-basiert und wurde

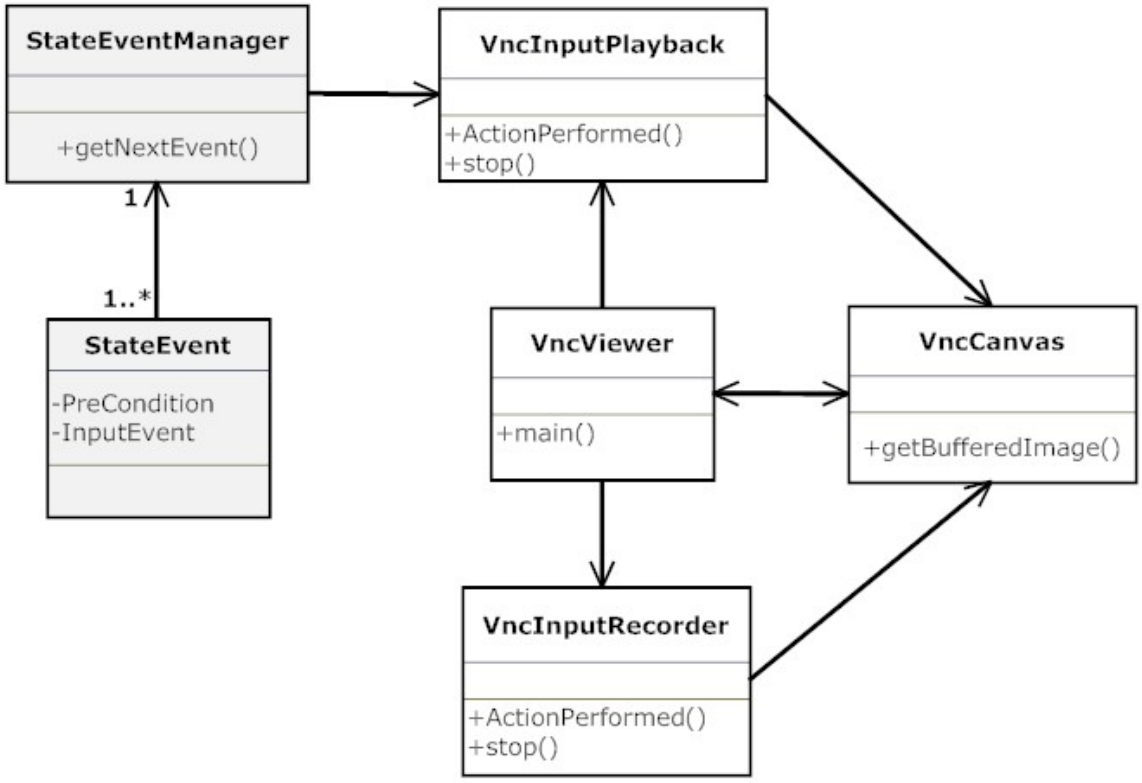


Abbildung 3.8: Modifizierte *VNCplay* Architektur

mit möglich wenigen Zuständen definiert, um deren Verwaltung zu vereinfachen. Die Abbildung 3.8 zeigt einen Ausschnitt des modifizierten *VNCplay*-Klassendiagramms. Die grün markierten Klassen stellen die neuen eingeführten Klassen dar. Die Klasse *StateEventManager* verwaltet das Triggern des Automaten und die Abbildung 3.9 stellt ein vereinfachtes Zustandsdiagramm des Automaten dar wie es in der Klasse implementiert wird.
Die Aufnahmephase verläuft wie es in der ursprünglichen *VNCplay* definiert wurde (hierzu s. [21]). Beim Start der Wiedergabephase wird ein erstes Mal die *Log*-Datei durchgegangen um den Automat aufzubauen. Die Automat-Struktur wird dann im Cache behalten. Ein zweites Mal wird die *Log*-Datei durchgegangen, um die zu emulierenden Benutzereingaben abzulesen und den Automat im Cache zu triggern. Selbstverständlich erfordert des Automaten erfordert die Erfüllung gewisser Voraussetzungen (*preCondition*). Zum Beispiel bevor ein Klick-Event getätigt wird, muss es eine Übereinstimmung zwischen den gespeicherten Pixel-Werten (nun definiert als *preCondition* im Automat) in der *Log*-Datei und in der emulierten Umgebung geben. Jeder Zustand im Automat hat einen Vorgänger und einen Nachfolger. Dies

vereinfacht das Zurückspringen zum vorherigen Zustand im Falle einer nicht erfüllten Vorbedingung (*preCondition*). Die Zustände im Automat sind von zwei Typen:
***PressEvent***: enthält Event wie *Mouse-*, *KeyPressEvent*, *Mouse-*, *ReleaseEvent* und *MouseDraggedEvent*. Sie besitzen eine Vorbedingung (*Sync-Point*), die erfüllt werden muss, bevor sie ausgeführt werden.
***MoveEvent***: Das sind Event wie *MouseMoveEvent*. Die Zustände von diesem Typ benötigen keine Vorbedingung. Sie werden ohne weitere Prüfung eingeschaltet und ausgeführt, denn die Ausführung dieser Events führt zu keiner Bildschirminhalt-Änderung.
Sollte die Vorbedingung eines Zustands vom Typ *PressEvent* nicht erfüllt werden, wird auf den vorherigen Zustand-Typ zurückgesprungen, der ein Zustand vom Typ *MoveEvent* sein muss. Die Abbildung 3.9 zeigt eine vereinfachte Darstellung des Automaten.
Die *Log*-Datei besteht aus einer alternierenden Folge zwischen dem Zustand-Typ *MoveEvent* und dem Zustand-Typ *PressEvent*. Die enthaltenen *Sync-Point* dienen als *preCondition* für Zustände des Typs *PressEvent*. Im Gegensatz zum ersten modifi-

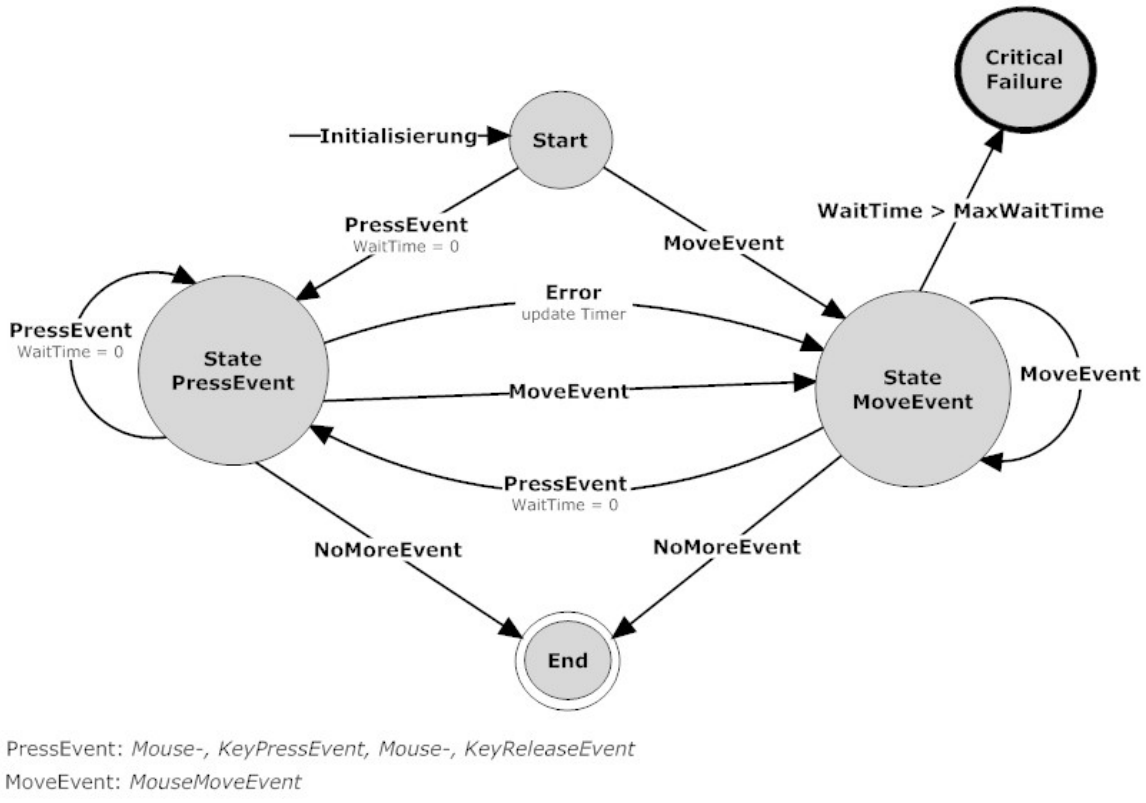


Abbildung 3.9: Vereinfaches Zustanddiagramm des Automaten

zierten *VNCplay* werden hier die *MouseMoveEvent* in der *log*-Datei mitgespeichert. Daher hat der Benutzer während der Wiedergabe nicht das Gefühl, dass die Maus von einem Punkt zu einem anderen springt.
Außerdem wird hier im Gegensatz zum [21] und zum [25] kein Pixel-per-Pixel Vergleich verwendet, um die *Sync-Point* zu vergleichen, sondern es wird ein Pattern-Vergleich angewandt. Der Pattern-Vergleich ist effizienter und einfacher zu implementieren. Außerdem ist er robuster als der Pixel-per-Pixel Vergleich, da nur die Muster von bedeutung sind. Somit werden auftretende Pixel-Fehler nicht betrachtet. Durch den Fehler-Toleranz-Grad kann je nach Bedarf eingestellt werden wie unterschiedlich zwei *Sync-Point* sein dürfen. Es wird dabei die Methode der JAI[6]-API verwendet. Die Funktionsweise des Pattern-Vergleichs ist ebenso trivial, es werden aus den zwei zu vergleichenden Bildern zwei Vektoren berechnet. Nun muss die Distanz zwischen diesen berechnet werden; diese entspricht der Differenz zwischen den Bildern. Bei einer Distanz gleich `0` sind die zwei Vektoren ähnlich. Ein ähnliches Beispiel dieses Ansatzes kann im [26] entnommen werden.

### Auswahlkriterien geeigneter Emulatoren

Ein Ziel der vorliegenden Arbeit ist die Entwicklung einer webbasierten Anwendung zur Überprüfung von aktualisierten Emulatoren anhand geeigneter Tests. Das zu entwickelnde System soll die Langzeitarchivierung digitaler Objekte unterstützen. Auf gegebenen Funktionalitäten sollen die Emulatoren überprüft werden. Der Emulator soll in der Lage sein, u.a. (veraltete) Systeme auszuführen bzw zu emulieren. Außerdem soll ein Remote-Zugriff auf das ausgeführte System möglich sein. Der Remote-Zugriff ist entscheidend für die automatische Durchführung von Testfällen.
Folgende Eigenschaften muss der getestete Emulator erfüllen:

- VNC-fähig sein: der ausgewählte Emulator muss die VNC-Schnittstelle bereitstellen können. Das ist eine der wichtigsten Eigenschaften, die der ausgewählte Emulator haben muss. Ohne diese Schnittstelle wird die Verwendung des Frameworks erheblich erschwert. Es besteht keine Möglichkeit mehr die Testfälle per Netzwerk auszuführen oder zu erstellen. Das *VNCplay* wird nicht mehr verwendbar sein und die Testautomatisierung wird dadurch ebenfalls nicht mehr durchführbar sein.

- Flexibilität: VNC-fähige Emulator bietet ein hohes Maß an Flexibilität: Plattform-Unabhängigkeit, das Framework ist über Internet aufrufbar und somit für alle Internet-fähigen Geräte auch aufrufbar.

[6] Java Advanced Imaging, Homepage http://java.sun.com/javase/technologies/desktop/media/jai

- Der Emulator soll auf modernen Systemen ausführbar sein.
- Ein *Open Source* Emulator ist wünschenswert aber nicht notwendig, denn dadurch dürfte der Emulator um eigene Funktionalitäten erweitert werden. Ein Beispiel hierfür zeigt die Erweiterung des *Open Source* Emulatoren Dioscuri(Version 0.7.0)[7] um eine VNC-Schnittstelle. Das Projekt wurde vom Lehrstuhl für Kommunikationssysteme der Albert-Ludwig-Universität Freiburg begleitet. Somit wäre der Emulator Dioscuri ein möglicher Kandidat, der im Emulator-Testing-Framework integriert sein könnte.
- Ein reichlich dokumentierter Emulator vereinfacht seine Verwendung und hilft hauptsächlich dem Entwickler, die Software zu verstehen.
- Die Haltbarkeit, das ist eine der Funktionalitäten, die ein Emulator im Allgemein haben muss[15].

[7] http://dioscuri.sourceforge.net/

# 4 Umsetzung

In diesem Kapitel wird nun die Implementierung des Emulatoren-Testing-Frameworks vorgestellt. Dabei geht es um Umsetzung des im vorherigen Kapitel beschriebenen Konzepts. Die Implementierung wird sich hauptsächlich auf die Erstellung und die automatische Durchführung von Testfällen konzentrieren. Die Implementierung der Softwarearchiv-Verwaltung wird aus [23] übernommen, daher wird hier nur teilweise behandelt.
Nun soll ein Blick auf den technischen Aspekt des Emulator-Testing-Frameworks geworfen werden. Bevor die konkrete Umsetzung behandelt wird, wird zuerst das prozedurale Ablaufdiagramm des Frameworks vorgestellt und dann wird das UML-Klassendiagramm beschrieben.

## 4.1 Ablaufdiagramm

Die Abbildung 4.1 stellt das prozedurale Ablaufdiagramm des Emulatoren-Testing-Frameworks dar. Das Diagramm illustriert die Folge der Aufrufe der Prozeduren bei der Durchführung von Testfällen.
Wenn das Framework zur Durchführung von Testfällen gestartet wird, werden zuerst die ausgewählten Testfälle und der dazu passende Emulator vom System aufgerufen. Die Testfälle werden nacheinander in der vom Emulator bereitgestellten, emulierten Umgebung ausgeführt. Jedoch wird für jeden Testfall die emulierte Umgebung neu gestartet, denn die Testfälle werden vom System unabhängig behandelt. Die Wiedergabephase erfolgt wie es im 3.3.2 beschrieben wurde. Ein Testfall wird erfolgreich bewertet, wenn die passende *Log*-Datei komplett abgearbeitet wird. Die Ergebnisse der Tests werden in einer Protokoll-Datei gespeichert und werden dem Benutzer nach dem Testlauf zur Verfügung gestellt. In dieser *Log*-Datei kann der Benutzer die Ergebnisse einzelner Testfälle herausnehmen

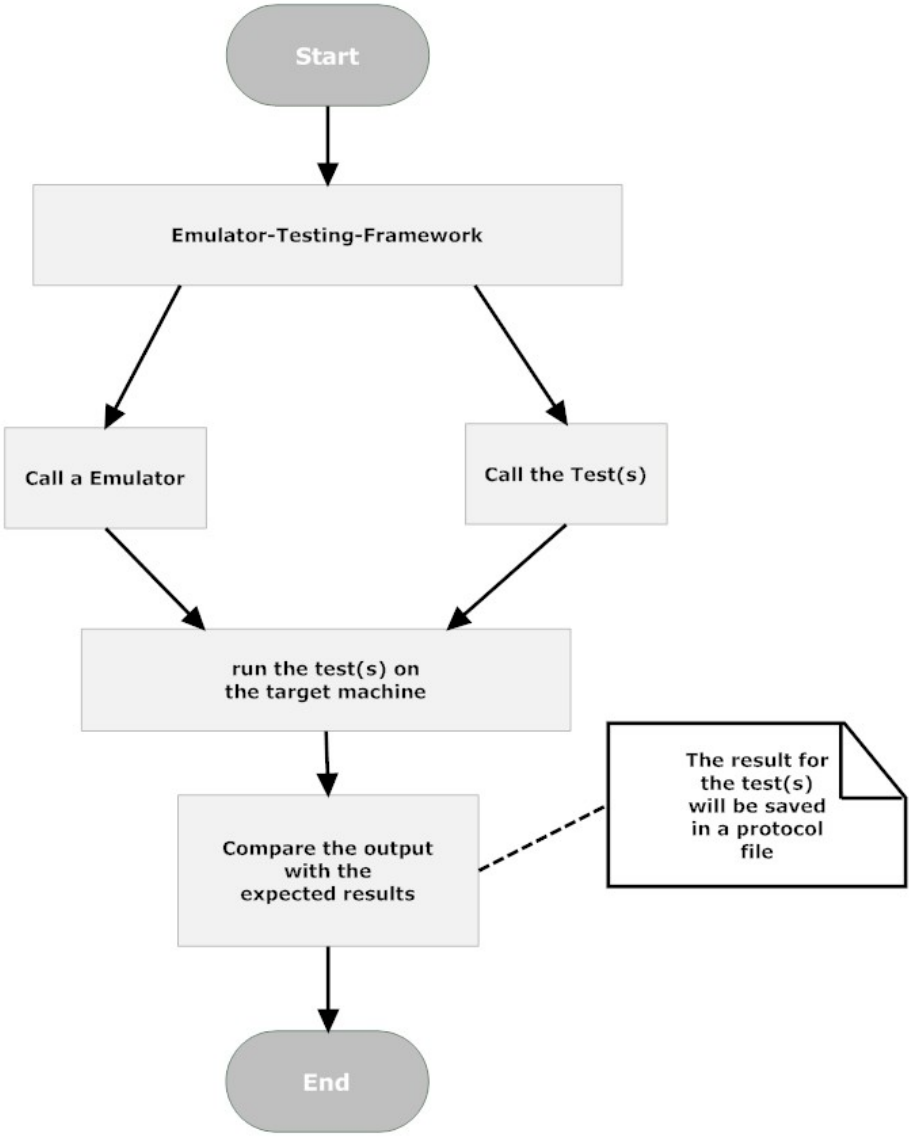


Abbildung 4.1: Prozedurales Ablaufdiagramm

## 4.2 UML-Klassendiagramm

Um einen sinnvollen Übergang von der abstrakten Idee zu einer konkreten Umsetzung zu ermöglichen, sollen zunächst die möglichen Arten von Beziehungen zwischen den Entitäten identifiziert werden. Die graphische Darstellung dieses Datenmodells wird in der Abbildung 4.2 als Klassendiagramm illustriert. Nachfolgend wird die einzelnen Entität vorgestellt:

**EmulatorTest**: stellt einen Testfall an sich selbst dar. Die Klasse enthält den Parameter *EmulatorType*, der festlegt, für welchen Emulator der Test konzipiert wurde. Diese Information ist für die Geschäftslogik von Bedeutung, denn in der Wiedergabephase soll der entsprechende Emulator aufgerufen und gestartet werden. Ein weiterer Parameter der Klasse ist *Image*. Dieser Parameter wird dem darunter liegenden *EmulatorType* zum Start der emulierten Umgebung übergeben.

**TestFile**: Für jede durchgeführte Aufnahme wird eine Binärdatei (*RecordingFile*) erstellt. Diese wird als Parameter der Klasse *TestFile* übergeben. Diese Datei wird mit einer eindeutigen ID identifiziert. Außerdem wird der entsprechende Test sprich *EmulatorTest* mit der Klasse *TestFile* verknüpft.

**Testprotokoll**: Zu jeder Instanz der Klasse *EmulotorTest* gehört ein *TestProtocol*. Diese Klasse modelliert die Protokoll-Datei, die nach jedem Testlauf dem Benutzer zur Verfügung gestellt wird.

**Image**: Es sind 3 Type zu unterscheiden: HDD-Image, Floppy-Image, CD_ROM-Image. Wenn dem Emulator ein Image als Paramater übergeben wird, wird anhand des Typs entschieden wie das Image dem Benutzer zur Verfügung gestellt wird. Der Typ eines Images wird vom Archivverwalter beim Einfügen des Images im Softwarearchiv festgelegt. Der Parameter *URL* lokalisiert das Image im Softwarearchiv.

**Emulator**: die Klasse stellt im Allgemeinen einen beliebigen Emulator in der Geschäftslogik dar. Für spezifische Emulatoren müssen allerdings die jeweiligen Klassen implementiert werden. Zum Beispiel aus der Abbildung 4.2 werden die Klassen *QEMU* und *Dioscuri* aus der Klasse *Emulator* abgeleitet. Diese Trennung ist notwendig, denn jeder Emulator hat eigene Spezifikationen und wird anders gestartet. Zum Beispiel wird der QEMU-Emulator mit dem Befehl `qemu -vnc :0 image.img` gestartet; der Emulator Dioscuri wird mit dem Befehl `java -jar dioscuri.jar` gestartet.

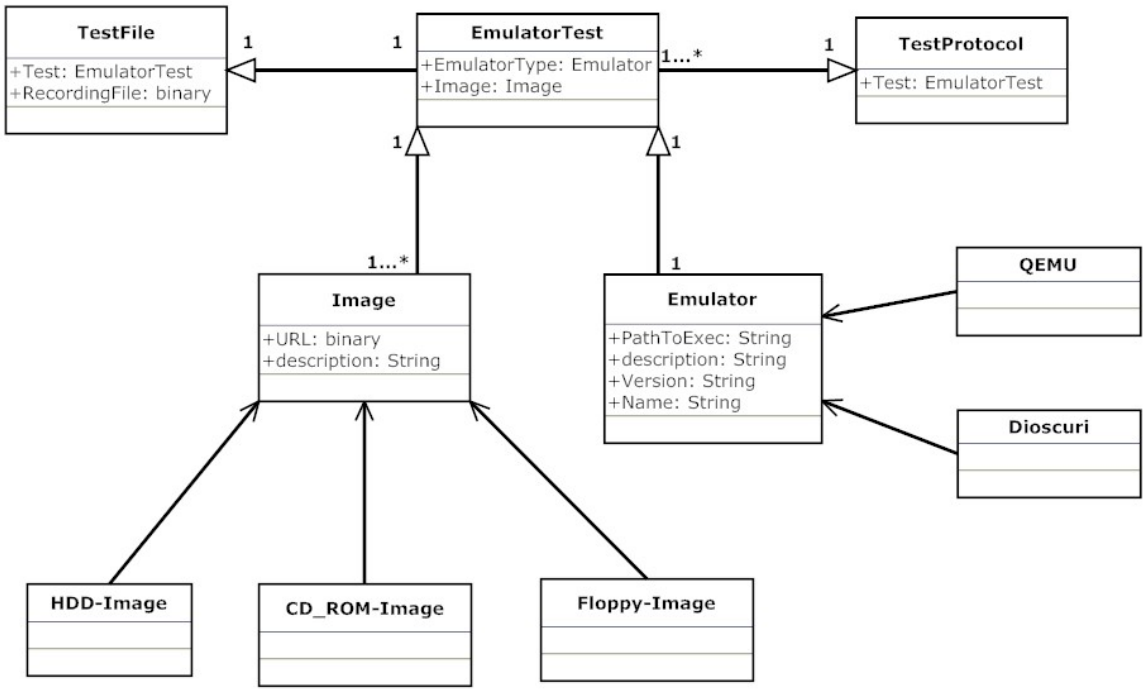


Abbildung 4.2: Emulator-Testing UML-Klassendiagramm

## 4.3 Anforderungen

Bevor im nächsten Abschnitt die konkrete Architektur des Prototyps vorgestellt werden kann, werden zuerst die Anforderungen erhoben. Nachfolgend sollen unter den verschiedenen Aktoren die einzelnen Anforderungen definiert werden:

### 4.3.1 Anforderungen des Benutzer oder Softwaretester

Das ist der Endbenutzer des Frameworks. Er erstellt und führt die Tests durch. Der Software-Tester arbeitet über ein Web-Interface. Ihm werden technische Details abstrahiert. Die Abbildung 4.3 zeigt einen Ausschnitt des Web-Interface zur Erstellung eines Testfalls für einen spezifischen Emulator (QEMU). Zuerst soll der Benutzer sämtliche Parameter zum Starten (Knopfdruck auf "*...to Recording*") des Emulators eingeben. Nachdem die emulierte Umgebung gestartet ist, kann die eigentliche Aufnahmephase (Erstellung vom Testfall) beginnen. Für die Testfall-Erstellung sind

allerdings Kenntnisse über das emulierte System notwendig. Zum Beispiel: das Starten einer Anwendung wie *Lotus Ami Pro* unter *Windows 3.11*. Die Abbildung 4.5 illustriert die Aufgaben der Benutzer in einem Use-Case.

- *Add Test*: Per Web-Interface wird dem Software-Tester die Möglichkeit gegeben Testfälle, zu erstellen
- *View Test*: Es wird die Liste der verfügbaren Tests angezeigt.
- *Run Test*: Aus den verfügbaren Tests kann der Benutzer eine Gruppe auswählen. Die ausgewählten Tests werden dann vom System automatisch nacheinander durchgeführt
- *Test Protocol*: Nach der Testdurchführung wird vom System eine Protokoll-Datei zurückgeliefert, die dem Benutzer zur Verfügung gestellt wird.
- *Edit Tests*: sollten die Parameter eines bereits angegebenen Test geändert werden, kann der der Benutzer dies editieren und wieder speichern.

Vom Anwender des Emulatoren-Testing-Frameworks wird nicht nur eine gewisse Kenntnis der emulierten Umgebung verlangt, sondern auch muss das System bzw. der Computer selbst, an dem der Anwender arbeitet, muss bestimmte Voraussetzungen erfüllen können.

#### Systemvoraussetzungen für den Anwender

Im Laufe der Entwicklung der Software wurde darauf geachtet, dass die Systemanforderungen auf der Benutzerseite minimal gehalten werden. Folgende Voraussetzungen sind für eine saubere Anwendung des Frameworks wünschenswert:

- PC mit einem modernen oder aktuellen Betriebssystem
- Sollte sich der Anwender außerhalb des lokalen Netzes befinden, wo der Applikation-Server bzw. Webserver ausgeführt wird, ist eine Verbindung zum Internet erforderlich.
- Der verwendete Webbrowser soll Java-fähig sein, um die Anzeige des *VNCplay* Applet zu ermöglichen.
- Eine JVM (Java Virtual Machine) soll auf dem Computer installiert sein. Aufgrund der in der Implementierung verwendeten Technologie wird eine Java-Version ab 1.4.2 empfohlen.

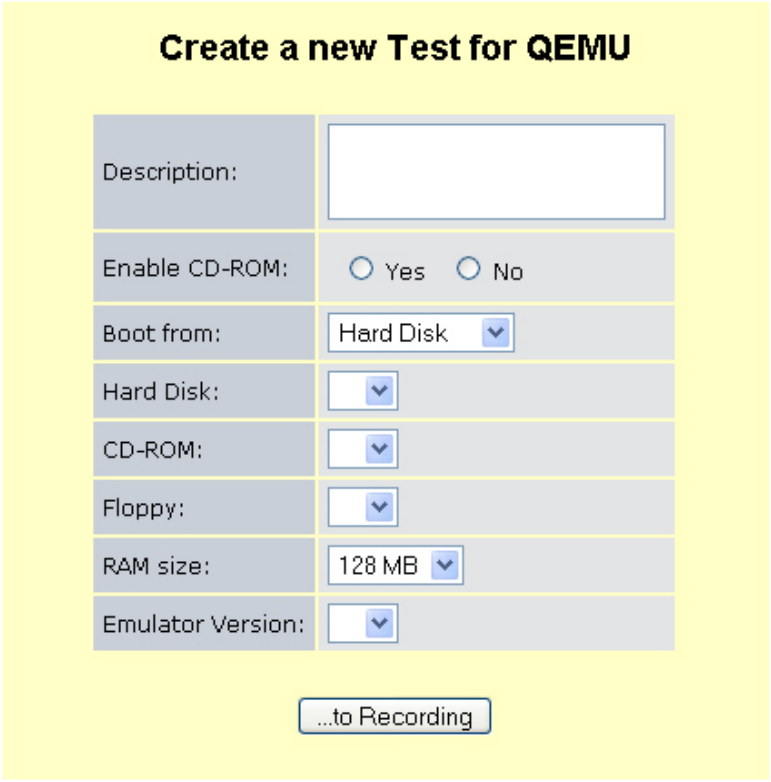


Abbildung 4.3: Web-Interface zur Erstellung eines neuen Tests

- Der Webbrowser soll ebenfalls die Ausführung von JavaScript zulassen.

- Standardmäßig lassen die modernen Webbrowser nicht die Ausführung von *Pop-Up*-Fernster zu. Dies soll aber aktiviert sein, um das eingebetteten Applet in der gerade angezeigten HTML-Seite erscheinen zu lassen. Ein Beispiel der Einbettung des *VNCplay* Applet wird in der Abbildung 4.4 illustriert.

```
<EMBED  type ="application/x-java-applet;version=1.6" width="700"
        height="500" pluginspage="http://java.sun.com/products/plugin/"
        java_code="VncViewer.class" java_codebase="."
        java_archive="VncplayEmuTest.jar" HOST="132.230.4.29" PORT="5900"
        autorecord="yes" mouse_accel="no"
        traceurl="http://132.230.4.29/SoftwareArchiveFrontend/recordings/createRecordFileFromTest.jsf"
        tracecookie="JSESSIONID=2784C1BFDAACDE87742F7C7863F89A5B" />
```

Abbildung 4.4: HTML Code zur Einbettung des *VNCplay* Applet

### 4.3.2 Anforderungen des Test-Managers

Die Aufgabe des Test-Managers und des Archivverwalters werden nicht getrennt sein, denn beide arbeiten daran, dem Endbenutzer des Frameworks eine funktionsfähige Anwendung zur Verfügung zu stellen. Der Test-Manager hat voll Zugriff auf das System. Er arbeitet direkt an der Server-Maschine (Hostmaschine), denn nur so wird es möglich sein, einen neuen Emulator im System hinzuzufügen (s. 4.5). Der Emulator soll auf der Hostmachine installiert sein und dies hängt von den lokalen Bibliotheken ab. Daher ist ein gut dokumentierter und ein einfach bedienbarer Emulator ein günstiger Ausgangspunkt, diese Aufgabe zu erfüllen. Außerdem soll der Test-Manager die benötigen Software und Komponente zur Erstellung und zur Durchführung von Testfällen zur Verfügung stellen. Dabei soll er beispielsweise Softwarekomponente neu kombinieren. Die Struktur und die Verwaltung des Softwarearchivs wurde aus [23] übernommen. Daher werden sie in dieser Arbeit nur zum Teil behandelt.
Ein Test-Manager kann selbstverständlich die Rolle eines Software-Testers übernehmen. Die Abbildung 4.5 schildert diese beiden Rolle in einem User-Case-Diagramm noch genauer:

#### Systemvoraussetzungen für den Test-Manager

Wie bei der Client-Maschine muss ebenfalls die Server-Maschine gewisse Systemvoraussetzungen nachweisen können:

- PC mit einem modernen oder aktuellen Betriebssystem (Ubuntu 10.04)
- Der Apache Webserver (Version ab 2.2.x)
- Ein JRE (*Java Runtime Environment*), Version ab 1.6

## 4.4 Architektur

Das Emulatoren-Testing-Framework ist eine Erweiterung eines schon an der Universität Freiburg im Rahmen des PLANETS-Projekts implementierten Frameworks, das selbst auf der GRATE-Architektur basiert ist. Die genaueren Spezifikationen und Beschreibung darüber können im [23] entnommen werden. Das hier implementierte Framework wird auf der *Open Source* Anwendung JBoss Application Server 6.0[1]

[1]Das ist meist die verwendete Java Application Server auf der Markt.

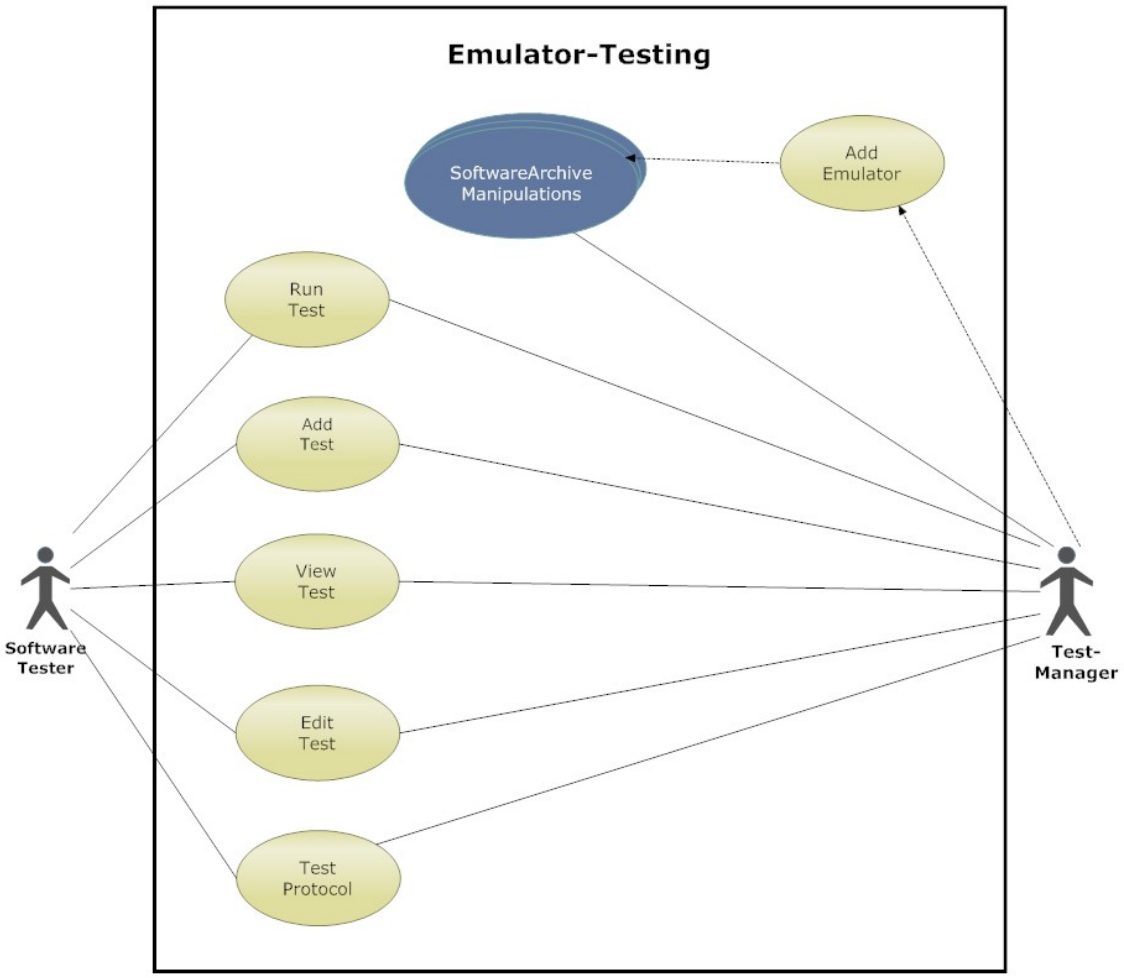


Abbildung 4.5: Use-Case des Emulator-Testing-Frameworks

(JBoos AS) aufgebaut, der die *Java EE 6*-Technologie[2] unterstützt. Außerdem macht das Framework Gebrauch vom *EJB 3.0*[3], das von *Java Enterprise Edition* angeboten wird. *EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology*[22].
Ein Ausschnitt des Emulatoren-Testing Architektur ist in der Abbildung 4.6 illustriert. Die Architektur kann in drei Schichten unterteilt werden und jede davon spielt eine ganz präzise Aufgabe.

[2]Java EE 6: Java Enterprise Edition 6 ist eine bedeutende Weiterentwicklung der im 1995 vom James Gosling erfundenen Programmiersprache Java
[3]Enterprise Java Beans

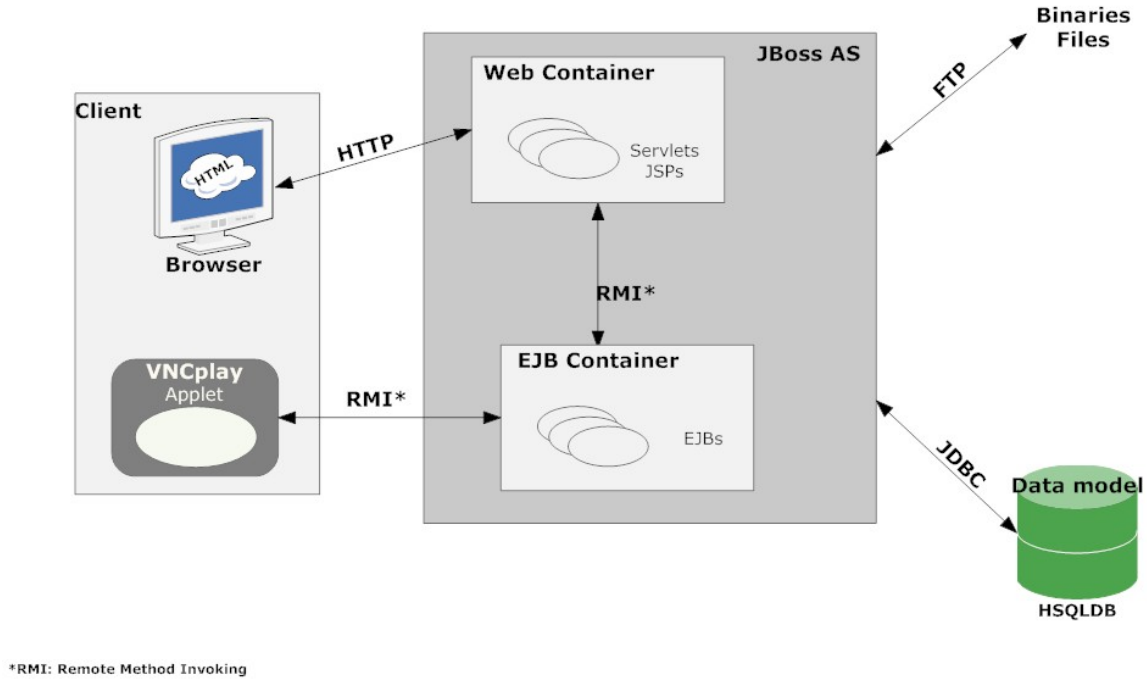


Abbildung 4.6: Emulator-Testing Architektur

### 4.4.1 Frontend

Das ist der sichtbare Teil der Anwendung. Er wird in einem Webbrowser angezeigt. Das Frontend abstrahiert die darunter liegenden technischen Details. Das ist die Schnittstelle zwischen dem Endbenutzer und dem System. Also kann der Benutzer durch das Frontend mit dem System interagieren. Alle Anfrage (HTTP[4], RMI[5]) vom Benutzer zum System findet auf dieser Schicht statt und wiederum alle Antworten des Systems werden auf dieser Schicht dargestellt. Die vom System registrierten Events während der Test-Erstellung werden auf diese Schicht initiiert.
Ein Auschnitt des Frontend wird in der Abbildung angezeigt.

### 4.4.2 Backend

Diese Schicht wird dem Benutzter abstrahiert. Das ist das Herz der Anwendung, denn hier wird die Geschäftslogik der Anwendung implementiert. In dieser Schicht werden

[4]Das Hypertext Transfer Protocol verwendet für die Übertragung von Daten in einem Netzwerk.
[5]Das Remote Method Invocation ist ein Java-API verwendet um entfernten Methode aufzurufen.

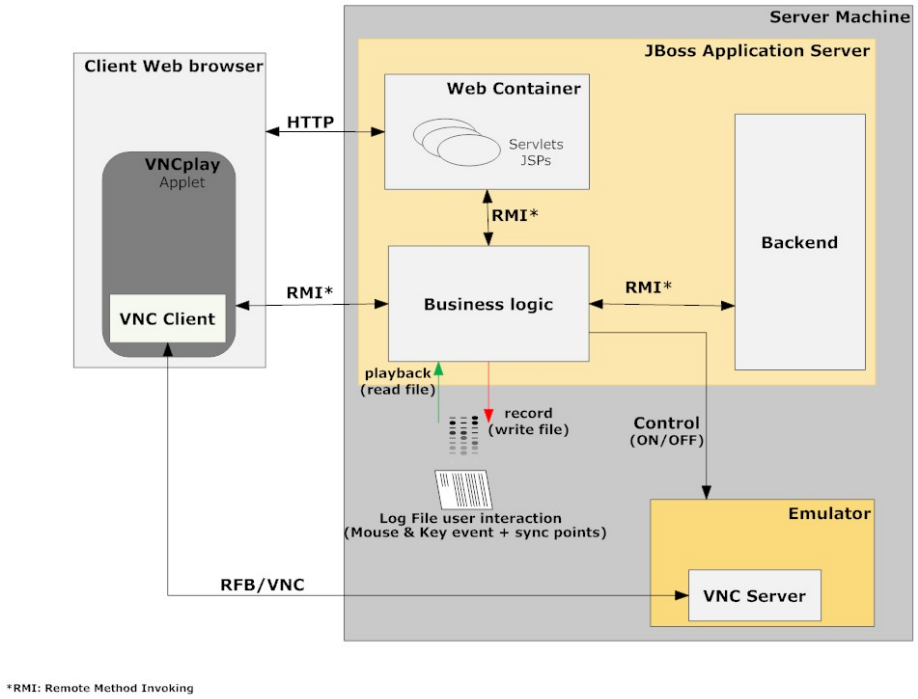


Abbildung 4.7: Emulatoren-Testing-Framework Frontend

die Regeln für die Antworten der Abfragen aus der Frontend definiert. Dazu gehören folgende APIs:

- Die *entity beans* deren Persistenz von der Mapping-Schicht gesichert wird
- *stateless* und *statefull beans*, die Methode zur Manipulierung der *entitity beans* bereistellen
- API JNDI, die Schnittstelle für Namensdienste und Verzeichnisdienste. Die dabei verwendete Protokoll ist RMI (Remote Method Invocation).

### Mapping-Schicht

In dieser Schicht wird das JPA[6] verwendet, um die physikalischen Daten in Objekte umzuwandeln und umgekehrt. Das Protokoll JDBC[7] wird dabei verwendet um die SQL-Anfragen auszuführen. Dies wird von der darunter liegenden relationalen Datenbank unterstützt.

[6] *Java Persistence API*: Schnittstelle für die Vereinfachung der Datenbankeneinträge.
[7] Das *Java Database Connectivity* ist eine Schnittstelle zu Datenbanken.

### Persistenzschicht

Diese Schicht stellt die eigentlichen gespeicherten Daten zur Verfügung. Die relationale Datenbank wurde als Persistenzschicht verwendet. Als Anfrage-Sprache für das darunter liegende Datenmodell wurde SQL (Structured Query Language) verwendet. Die Syntax von SQL ist relativ einfach aufgebaut und außerdem wird sie von fast allen gängigen Datenbanksystemen unterstützt.

# 5 Experiment

Nach der Umsetzung des Konzepts sollen nun die Tests erfolgen. Die durchgeführten Tests sollen zeigen, dass das Emulatoren-Testing-Framework seine Spezifikationen und seine Aufgabe erfüllt. Zu diesem Zweck wurden triviale Testszenarien durchgeführt wie z.B. ein Betriebssystem booten, eine Anwendung installieren und starten. Bevor die durchgeführten Tests beschrieben werden, sollen erstmals die Test-Umgebung und die verschiedenen vorgenommenen Einstellungen erklärt werden.

## 5.1 Einstellung

Als Server Maschine wurde ein Rechner mit einer öffentlichen IP-Adresse ausgewählt, um die Durchführung und die Erstellung der Testfälle über Internet zu ermöglichen. Darüberhinaus erfüllen die eingesetzten Maschinen, sowohl Server als auch Client die Auswahlkriterien, die im Kapitel 3.3 aufgelistet wurden. Die Topologie der Test-Umgebung wird in der Abbildung 5.1 dargestellt. In der Aufnahmephase werden die Benutzereingaben protokolliert und zugleich in eine *Log*-Datei geschrieben. Der Benutzer empfängt hingegen die Framebuffer Daten, um stets den Stand des Bildschirminhalts der emulierten Umgebung nachzuvollziehen. Somit werden die *Mouse* und *Key Event* wie gewünscht getätigt. Bei Durchführung der Testfälle ist der Benutzer eher ein Zuschauer, denn er hat keinen Zugriff auf das Geschehen in der emulierten Umgebung. Er empfängt weiterhin die Frame Buffer, um den Verlauf der Testfälle besser nachzuvollziehen.

### 5.1.1 Emulatoren

Zur Auswahl der zum Emulatoren-Testing-Framework passenden Emulatoren müssen einige Voraussetzungen erfüllt werden. Anhand der zur Verfügung stehenden Mittel konnte sich die Auswahl nur auf freie erhältliche Software erstrecken. Daher wurden QEMU und Dioscuri ausgewählt.

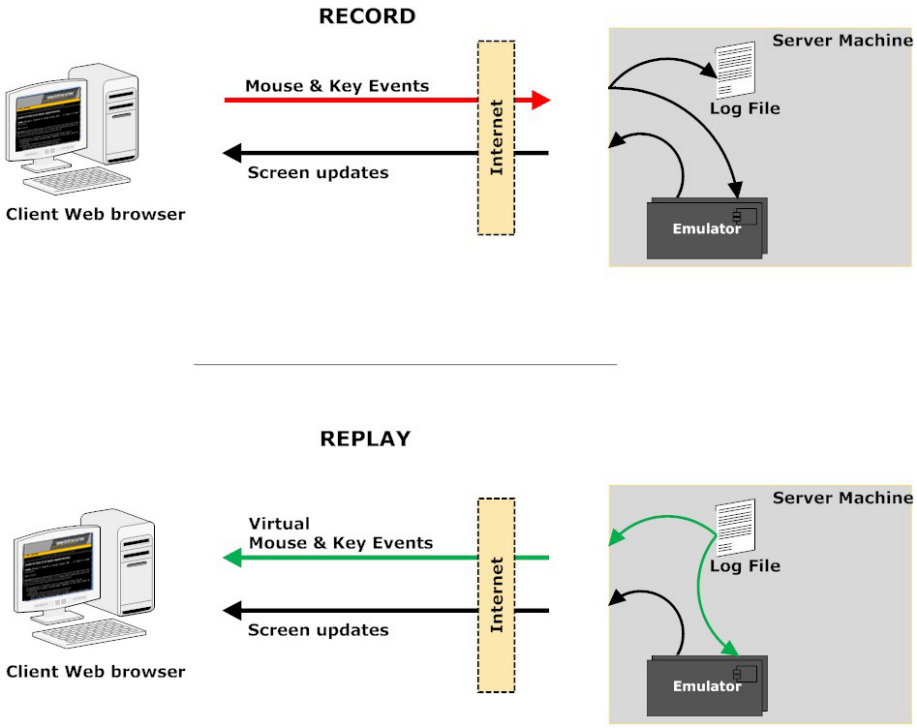


Abbildung 5.1: Struktur der Test-Umgebung

**QEMU**: die freie virtuelle Maschine ist seit 2003 erhältlich und wurde von Fabrice Bellard implementiert. QEMU ist in der Lage x86-Architektur zu emulieren. Insbesondere: PPC, ARM und SPARC. Der Emulator verfügt ebenfalls über eine VNC Schnittstelle, die den Zugriff zur emulierten Umgebung ermöglicht. Die gut dokumentierte Software hat die Erstellung und die Durchführung von Testfällen vereinfacht. Folgende QEMU-Versionen wurde verwendet:

- qemu-10.6
- qemu-11.1
- qemu-12.4
- qemu-12.5
- qemu-13.0
- qemu-14.0

**Dioscuri**: ein Java-basierter Emulator, der eine modulare Struktur hat. Jedes Modul emuliert die Funktionalität eines Hardware-Komponente (Prozessor, Speicher, Tastatur). Bei Emulation einer Plattform müssen lediglich die benötigen Modulen kompiliert bzw. geladen werden. Somit wird die Konfiguration des Emulators einfacher. Mit der modularen Struktur kann ein sehr hoher effizienter und zuverlässiger Emulationsgrad erzielt werden. Dioscuri ist in der Lage eine x86-Architektur zu emulieren und ist Plattform-unabhängig. Der Emulator wurde für die digitale Archivierung implementiert und wurde von der *Koninklijke Bibliotheek*[1] und dem *National Library of the Netherlands*[2] unterstützt. Da die früheren Versionen von Dioscuri keine VNC Schnittstelle haben, wurde lediglich die aktuellste Version betrachtet:

- dioscuri-0.7

### 5.1.2 Hostmachine

Der als Hostmaschine verwendete Computer hat folgende Eigenschaften:

- Ubuntu 10.04 LTS (April 2010)
- JBoss AS-6.0.0.20100911-M5 wurde als Webserver für das Framework verwendet. Der Service horcht den Port `8080`.
- Apache Server 2.2 wurde als Port-Weiterleitung verwendet, so dass alle am Port `80` ankommenden HTTP-Anfrage, wurde an den `8080` (JBoss AS) weitergeleitet.. Anstatt der Adresse: `http://hostmachine:8080/context` wird `http://hostmachine/context` zum Aufruf des Frameworks verwendet.
- Java Runtime 1.6.

### 5.1.3 Betriebssymsysteme & Software

In diesem Unterabschnitt sollen die zum Test verwendeten Betriebssysteme und Anwendungen aufgelistet werden.
**Betriebssysteme**: Windows 3.11, FreeDos, Windows 98, Windows XP.
**Software**: Lotus Amipro.
Nun ist die Test-Umgebung eingerichtet, die eigentlichen Tests können durchgeführt werden.

[1] Siehe hierzu: http://www.kb.nl/hrd/dd/dd_projecten/projecten_emulatieproject-en.html
[2] Home page http://www.nationaalarchief.nl

## 5.2 Testfälle

Die meist durchgeführten Testszenarien wurden mit QEMU absolviert, denn der Dioscuri mit seiner aktuell implementierten VNC Schnittstelle nicht die genügende Stabilität aufweisen konnte. Außerdem gab es keine Möglichkeit, Dioscuri per Kommandozeile im VNC-Modus zu starten. Mit der Kommandozeile wird dem System die Steuerung des Prozesses überlassen. Somit wird die Verwaltung der automatischen Durchführung der Testfälle und der zur Verfügung stehenden VNC Ports vereinfacht.
**A - Das Betriebssystem Windows 3.11 aus einer Festplatte booten**: Über

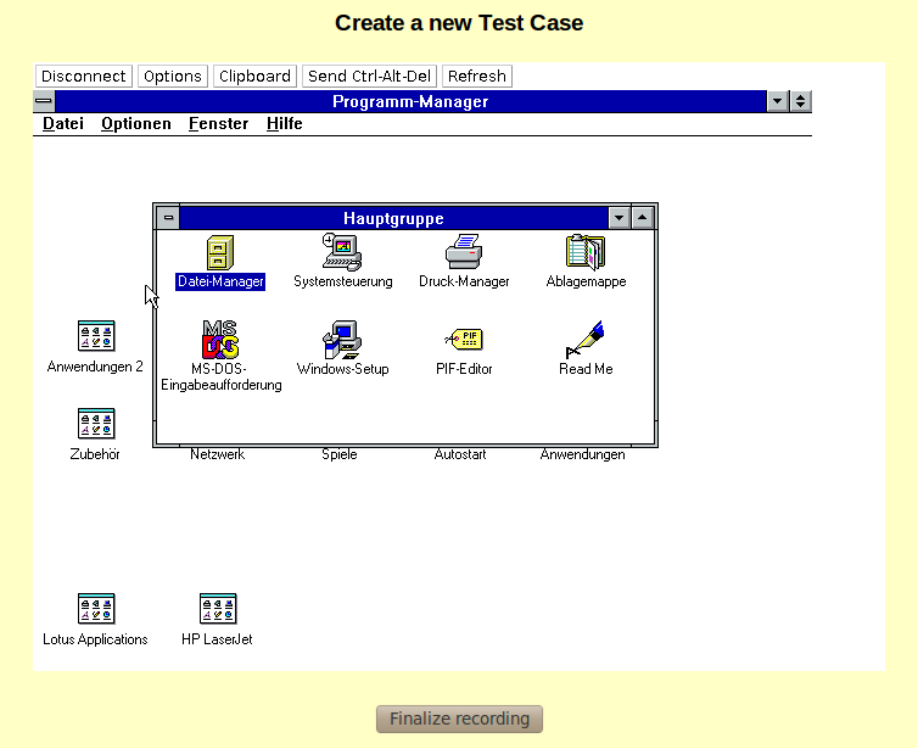


Abbildung 5.2: Windows 3.11 unter Ubuntu 10.4

das Web-Interface kann das Windows 3.11 Image als Festplatte ausgewählt und die Boot-Option auf *"Hard Disk"* gesetzt werden. Die Wahl vom Floppy und CD-ROM ist in diesem Testszenario nicht relevant. Als Emulator wurde der QEMU (qemu-0.10.6) ausgewählt. Nach dem Klick auf dem Knopf *"...to Recording"* kann die Aufnahmephase bzw die Erstellung des Testfalls beginnen. Das Betriebssystem wird ordnungsmäßig gestartet und es wird ein paar Maus-Klicks getätigt, die als Referenzpunkte während der Wiedergabephase dienen sollen. Nach Klick auf dem Knopf *"disconnect"* und anschließend auf *"finalize recording"* wird die entsprechende *Log-*

Datei erstellt und gespeichert.
**B - Die Anwendung *Lotus Amipro* in Windows 3.11 starten**: wie im vorgherigen Testfall beschrieben ist, wird das Betriebssystem Windows 3.11 gestartet und unter "Datei-Manager"die Anwendung *Lotus Amipro* gestartet. Danach kann des Testfall genauso erstellt und gespeichert werden.
**C - Die Anwendung *Lotus Amipro* in Windows 3.11 installieren**: bei der Paramatereingabe soll zusätzlich ein CD-ROM mit der zu installierenden Software auswählen. Nach dem Start des Betriebssystems soll die Installation gestartet werden. Die einzelnen Testschritte zur Installation müssen verfolgt werden. Dabei werden die die nötigen *Sync Point* erstellt und in der *Log*-Datei geschrieben. Nach der erfolgreichen Installation wird anschließlich die *Log*-Datei gespeichert.

## 5.3 Ergebnisse

Die Durchführung der oben genannten Testfälle erfolgt über das freundliche User-Interface. Zu jedem ausgewählten Emulator wird die Liste der verfügbaren Testfälle angezeigt. Die Abbildung 5.3 zeigt die Liste der verfügbaren Tests für den Emulator QEMU. Es wird für jeden angezeigten Testfall die Möglichkeit gegeben, die entsprechende *Log*-Datei herunterzuladen und anzusehen. Die ausgewählten Testfälle kön-

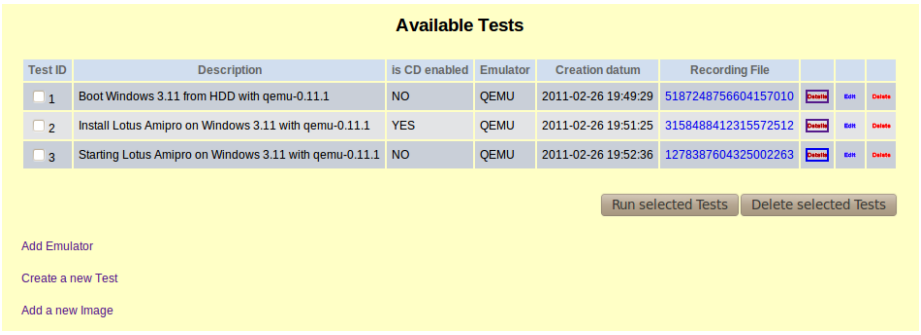


Abbildung 5.3: Testfall-Liste

nen vom Benutzer durchgeführt werden. Da die Testfälle in der Version *qemu-10.6* erstellt wurden, wurden sie nacheinander in der Versionen *qemu-11.1*, *qemu-12.4*, *qemu-12.5*, *qemu-13.0*, *qemu-14.0* durchgeführt. Die Testfälle schlugen in der Versionen *qemu-12.4* und *qemu-12.5* fehl. Aufgrund eines Bugs ist der Mauspointer in dieser Versionen per VNC nicht steuerbar. Für die übrigen Versionen sind die Tests-Ergebnisse positiv gewesen.

# 6 Zusammenfassung und Ausblick

Während die Langzeitarchivierung klassischer Objekte relativ trivial ist, bedarf die der digitalen Objekte genauerer Überlegungen, um die daraus resultierenden Herausforderungen zu bewältigen. Die vorliegende Arbeit setzt den Schwerpunkt auf die Emulation als Strategie der Langzeitarchivierung. Die Emulationsstrategie stellt die benötigen Komponenten zur Wiederherstellung der ursprünglichen Ablaufumgebung der digitalen Objekte virtuell bereit. Damit kann die Authentizität und die Integrität der digitalen Objekte gewährleistet werden. Welche Komponente zur Wiederherstellung der Ablaufumgebung eines gegebenen Objekts benötigt wird, hängt vom *View-Path* ab. Dabei werden die benötigen Komponente durch Software mit identischen Funktionen ersetzt.

Der Erfolg der Emulationsstrategie ist nicht vom eingesetzten Emulator zu trennen. Mit den technologischen Fortschritten sollen neue Hard- und Software-Landschaften eingesetzt. Die Emulatoren werden von Emulatoren-Hersteller dementsprechend aktualisiert. Zur Gewährleistung der Verlässlichkeit und der Zuverlässigkeit der in der Emulationsstrategie eingesetzten Emulatoren müssen bestimmte Maßnahmen ergriffen werden. Damit wird die Sichtbarmachung und die Bearbeitung digitaler Objekte weiter ermöglicht.

Das entwickelte Konzept sollte die Erstellung und die automatische Durchführung geeigneter Emulatoren-Tests ermöglichen. Die externen Funktionalitäten der Emulatoren waren dabei von Bedeutung. Das heißt zum Entwurf der Testfälle sind keine Kenntnisse der inneren Struktur des Emulators nötig. Daher wurde das *Black-Box*-Prinzip als Test-Strategie verwendet. Die im Emulatoren-Testing-Framework zum Einsatz kommenden Emulatoren mussten im Vorfeld bestimmte Voraussetzungen erfüllen. Eine solcher Voraussetzungen ist die Bereitstellung einer VNC Schnittstelle, welche eine zentrale Rolle in der Erstellung und der automatischen Durchführung von Testfällen spielt. Die automatische Test-Durchführung trägt zur Minimierung der Personalressourcen, der finanziellen Kosten und der Test-Laufzeit bei.

Zur Umsetzung des beschriebenen Konzepts wurde ein Ablaufdiagramm, das die Folge der Aufrufe der Prozeduren bei der Durchführung von Testfällen illustrierte,

vorgestellt. Dann wurden die möglichen Arten von Beziehungen zwischen die Entitäten durch ein UML-Klassendiagramm identifiziert. Danach wurden die Anforderungen der Anwender und der Test-Manager erhoben. Folgend wurde das Use-Case Diagramm, welche die Aufgabe einzelner Aktoren deutlicher machte, vorgestellt. Aus diesen Erkenntnissen wurde die Architektur des Emulatoren-Testing-Framework entwickelt. Das Framework bestand aus einem Backend und einem Frontend. Das Backend beschreibt die darunter liegende Geschäftslogik und das Frontend stellt den visuellen Teil dar. Durch das Frontend wird dem Benutzer die Möglichkeit gegeben, die Testfälle zu erstellen und sie automatisch durchzuführen.

Als Anwendung des implementierten Prototyps wurden zwei Emulatoren ausgewählt:

1. **Dioscuri** ist ein Java-basierter Emulator, der für die funktionale Archivierung entworfen wurde. Daher ist Dioscuri ein idealer Kandidat für das Emulatoren-Testing-Framework. Allerdings war die in der aktuellsten Dioscuri Version `0.7` eingebaute VNC Schnittstelle nicht ausreichend stabil, um eine Test-Erstellung und automatische Test-Durchführung zu unterstützen. Aus diesem Grund konnten für diesen Emulator keine aussagekräftigen Tests durchgeführt werden.

2. **QEMU** ist einer der meist verwendeten Emulatoren. Er kann viele Rechnerarchitekturen emulieren und ist deshalb sehr flexibel einsetzbar. Außerdem konnte der Emulator QEMU die aufgestellten Voraussetzungen erfüllen. Daher wurde QEMU im Emulatoren-Testing-Framework eingesetzt. Die Ergebnisse der durchgeführten Tests konnten belegen, dass einige QEMU Versionen an Funktionalität verloren haben. Allerdings sind diese Funktionalitäten bei anderen Versionen erhalten geblieben.

Die Zuverlässigkeit der in einem digitalen Archiv verwendeten Emulatoren kann sichergestellt werden, in dem die Emulatoren durch geeignete Tests untersucht werden. Das hier implementierte Emulatoren-Testing-Framework löst die eingangs vorgestellten Probleme und wäre für weitere Verbesserungen oder Erweiterungen offen:

- Bisher bedarf das Einfügen neuen Emulatoren ins Framework zwar einen minimalen Aufwand, aber dieser Vorgang könnte noch optimiert werden.

- Denkbar wäre auch eine Schnittstelle (z.B.: XML), mit der das Framework in andere Projekte innerhalb der digitalen Langzeitarchivierung eingebaut werden könnte. Beispielsweise das Open PLANETS-Projekt oder die momentanen an der Alberts-Ludwigs-Universität Freiburg laufenden Projekte im Bereich der Langzeitarchivierung[1].

[1] Siehe hierzu: http://www.ks.uni-freiburg.de/projekte/fla und http://www.ks.uni-freiburg.de/php_warbeit.php?akt=ueber1

- Die Wiedergabe einer aufgenommenen VNC-Session könnte noch effektiver implementiert werden; Die Erkennung der Zeichen oder der Tastenkombinationen könnte besser ausgehandelt werden. Während einer Wiedergabephase, bevor das erste Zeichen einer Kette angezeigt wird (z.B.: in einem Text-Editor), wird eine *Sync-Point* Übereinstimmung durchgeführt. Bei einer erfolgreichen Übereinstimmung gibt es allerdings keine Sicherheit, ob die angezeigten Zeichen aus der Kette den tatsächlichen Zeichen aus der *Log*-Datei entsprechen.

- Trotz der Schlankheit des verwendeten *VNCplay*, wurde bei der Übertragung von Bildern Verzögerungen registriert. Diese Verzögerungen könnten durch den Einsatz besseren Komprierungsalgorithmen minimiert werden.

- Sollte die VNC Schnittstelle stabiler werden, könnte der modulare Emulator Dioscuri, der von einem JVM abhängt, ein zukunftsfähiger Kandidat im Kontext der Langzeitarchivierung sein, insbesondere für das Emulatoren-Testing-Framework.

- Die automatische Überprüfung der Ton-Wiedergabe wäre auch eine denkbare Erweiterung für das Emulatoren-Testing-Framework.

# Literaturverzeichnis


**1** QEMU, Kernel-based Virtual Machine (KVM) and libvirt. http://qemu-buch.de/, 2010.

**2** M. N. Alam. Software Test Automation - Myths and Facts. http://www.benchmarkqa.com/pdf/papers_automation_myths.pdf, 2007.

**3** Bj Rollison Alan Page, Ken Johnson. How We Test at Microsoft. Preview: http://www.benchmarkqa.com/pdf/HowWeTest_Intro_ch14.pdf, 2008.

**4** Tilo Linz Andre Spillner. Basiswissen Softwaretest: Aus und Weitrbildung zum Certified Tester. 3. Auflage - Foundation Level, nach ISTQB-Standard, 2005.

**5** Fabrice Bellard. QEMU, a Fast and Portable Dynamic Translator. FREENIX Track: 2005 USENIX Annual Technical Conference, 2005.

**6** Stefan Rohde-Enslin Dirk Witthaut Andrea Zierer, Arno Dettmers. Digitalisierung und Erhalt von Digitalisaten in deutschen Museen. nestor - http://www.langzeitarchivierung.de, 2005.

**7** Jeffrey van der Hoeven Dr. Dirk von Suchodoletz. Emulation: From Digital Artefact to Remotely Rendered Environments. The International Journal of Digital Curation - Issue 3, Volume 4, 2009.

**8** Tobias Ott Dr. Gunnar Fuelle. Langzeiterhaltung digitaler Publikationen - Archivierung elektronischer Zeitschriften (E-Journals), 2006.

**9** Drs. Johan F. Steenbergen Dr. Raymond J. Van Diessen. The Long-Term Preservation Study of the DNEP Project. http://www.kb.nl/hrd/dd/dd_onderzoek/reports/1-overview.pdf, 2002.

**10** Reichherzer T. & Brown G. Quantifying software requirements for supporting archived office documents using emulation. Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries, International Conference on Digital Libraries (ICDL). USA: Chapel Hill, 2006.

**11** Anne Mette Jonassen Hass. Guide to Advanced Software Testing, 2008.

**12** Lampe Clifford Hedstrom Margaret. Emulation vs. Migration: Do Users care?, 2001.

**13** Jamie L. Mitchell Beth Anderson Chuck LeCount Jan Bagley Joy Islam. Testability Engineering. http://www.benchmarkqa.com/pdf/papers_testability_engineering.pdf, 2007.

**14** Rothenberg Jeff. Ensuring the Longevity of Digital Information. http://www.clir.org/pubs/archives/ensuring.pdf, 1999.

**15** Bram Lohman Jeffrey van der Hoeven. Building the emulator - A behind-the-scene look at developmen. Koninklijke Bibliotheek (KB), 2006.

**16** Hilde van Wijngaarden Jeffrey van der Hoeven. Modular emulation as a long-term preservation strategy for digital objects. Koninklijke Bibliotheek, the National Library of the Netherlands The Hague, The Netherlands, 2005.

**17** Gregory M. Kapfhammer. Software Testing. Department of Computer Science Allegheny College, 2010.

**18** Randolph Welte Klaus Rechert, Dirk von Suchodoletz. Emulation Based Services in Digital Preservation. Proceedings of the 10th annual joint conference on Digital libraries, 2010.

**19** Oak Ridge National Laboratory. Pentium Floating Point Division Bug. Radiation Shiedling Information Center (RSIC) Newsletter, December, 1994.

**20** Sheon Montgomery Margaret Hedstrom. Digital Preservation Needs and Requirements in RLG Member Institutions. A study commissioned by the Research Libraries Group - http://www.oclc.org/research/activities/past/rlg/digpresneeds/digpres.pdf, 1998.

**21** Ramesh Chandra Nickolai Zeldovich. Interactive Performance Measurement with VNCplay. http://suif.stanford.edu/vncplay/, 2005.

**22** ORACLE. Enterprise JavaBeans Technology. http://www.oracle.com/technetwork/java/index-jsp-140203.html, 2011.

**23** Mario Philipps. Entwurf und Implementierung eines Softwarearchivs für die digitale Langzeitarchivierung. Dissertation zur Erlangung des Grades Diplom Inf. der Fakultät für Angewandte Wissenschaften der Albert-Ludwigs-Universität Freiburg im Breisgau, 2010.

**24** Tristan Richardson. The RFB Protocoll - Version 3.8, last updated 26 November 2010. RealVNC (formely of Oliverty Reseach Ltd / AT&T Labs Cambridge), 2010.

**25** Felix Ruzzoli. Ein Framework für die zustandsbasierte Fehlererkennung und -behandlung von interaktiven Arbeitsabläufen. Dissertation zur Erlangung des Grades BSc. der Fakultät für Angewandte Wissenschaften der Albert-Ludwigs-Universität Freiburg im Breisgau, 2010.

**26** Rafael Santos. Java Image Processing Cookbook. http://www.lac.inpe.br/JIPCookbook/, 2010.

**27** Granger Stewart. Emulation as a Digital Preservation Strategy, 2000.

**28** Dr Raymond J. van Diessen. Preservation requirements in a deposit system. Technical Report, IBM / KB Long-term Preservation Study, 2002.

**29** Erik van Veenendaal. Standard glossary of terms used in Software Testing. Version 2.1. International Software Testing Qualifications Board (ISTQB), 2010.

**30** Hoeven van der J.R. Verdegem R. Emulation: to be or not to be. The Archiving 2006 Final Program and Proceedings. IS&T Archiving Conference. Canada: Ottawa, 2006.

**31** Dr. Dirk von Suchodoletz. Erfolgsbedingungen des Einsatzes von Emulationsstrategien, 2008.

**32** Klaus Rechert Dr. Dirk von Suchodoletz Randolph Welte Maurice van den Dobbelsteen Bill Roberts Jeffrey van der Hoeven Jasper Schroder. Novel Workflows for Abstract Handling of Complex Interaction Processes in Digital Preservation, 2009.

**33** Randolph Welte. Funktionale Langzeitarchivierung digitaler Objekte - Entwicklung eines Demonstrators zur Internetnutzung emulierter Ablaufumgebungen. Dissertation zur Erlangung des Doktorgrades der Fakultät für Angewandte Wissenschaften der Albert-Ludwigs-Universität Freiburg im Breisgau, 2008.

**34** Laurie Williams. Testing Overview and Black-Box Testing Techniques, 2006.

# Abbildungsverzeichnis

# CD-Inhalt

Auf der CD-ROM befinden sich die folgenden Ordner/Dateien:

- **Masterarbeit**: sie enthält eine weitere Verfassung dieser Arbeit im PDF-Format.
- **SoftwareArchive**: das EAR-Projekt.
- **SoftwareArchiveBackend**: ein Java-Projekt, das den Backend-Code enthält
- **SoftwareArchiveBeans**: die in der Geschäftslogik verwendete *stateful* und *stateless* Session Beans werden in diesem EJB-Projekt beschrieben.
- **SoftwareArchiveBeansClient**: das lokale Interface für die in SoftwareArchiveBeans implementierten Beans
- **SoftwareArchiveFrontend**: der visuelle Teil des Emulatoren-Testing-Frameworks wird in diesem JSF-Projekt implementiert.
- **VncplayEmuTest**: diese Ordner enthält den Code des modifizierten *VNCplay*.
- **Lib**: enthält alle benötigen Bibliotheken bzw. Jar-Dateien zum Starten der Anwendung.
- **README.txt**: Enthält diese Beschreibung.
- **Softwarearchiv.userlibrairies**: die in Eclipse einzubindenden Bibliotheken.

**Versionen:**

- Java 1.6
- Dynamic web Module 2.5
- JBoss-6.0.0.20100911-M5
- EJB Module 3.0
- EAR 5.0

Das Emulatoren-Testting-Framework ist über ein *git*-Repository verfügbar. Zum *checkout* ist folgendes *git*-Befehle nötigt: `git clone 132.230.4.29:/srv/svn`

# Index